# Verifying Quantum Communication Protocols with Ground Bisimulation

**Xudong Qin**
Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China
marsxd@gmail.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

**Related Works.** The problem of automated equivalence checking was also noted by Ardeshir-Larijani et al. [1]. They introduced stabilizer formalism and restrict themselves to certain operators called Clifford operators to enable them be able to verify the protocols with arbitrary inputs. The quantum protocols behave functionally so that its equivalence checking could be considered as computing a deterministic input-output relation for all interleavings of the concurrent system. These relations can be abstracted by superoperators that can be extended to arbitrary inputs according to their linearity. Moreover, the introduction of the stabilizer formalism avoids the problem of explosion of states arising from branching and cocurrency in the programs. The usual state vector description of a $n - qubits$ quantum state requires $2^n$ amplitudes to be specified. While in stabilizer formalism, quantum state can be described in $O(n)$ space using a tableau and each quantum operation applied on it can be simulated in $O(n^2)$ time.

In the same paper [1], Ardeshir-Larijani et al. proposed $CCS^q$, a concurrent language for describing quantum systems similar with $qCCS$, to describe the specification and implementation of quantum protocols. It has well-defined operational semantics and superoperator semantics for those protocols which behave functionally. Basing on the $CCS^q$, they implemented a tool called QEC (Quantum Equivalence Checker) to verify the correctness of quantum protocols in exponential time, ranging from quantum communication protocols, error-correction protocols, teleportation protocols to fault-tolerant protocols. However, it has not reported the experiment about the verification of cryptographic protocols such as BB84.

Besides bisimulation, model-checking is another feasible approach verifying the correctness of quantum programs. As an illustration, Feng et al. [11] proposed an quantum extension of


PCTL and developed a quantum model-checking algorithm basing on the quantum Markov Chains. The algorithm is used for model-checking the correctness of the basic BB84 protocol as a demonstration.

## 2 Preliminaries

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 $\Delta$ 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 $\lceil \overline{s} \rceil = \{s\}$. In this paper we only need to use distributions with finite support, and let $Dist(S)$ denote the set of finite support distributions over $S$, ranged over by $\Delta, \Theta$ etc. If $\sum_{k\in K} p_k = 1$ for some collection of $p_k \geq 0$, and the $\Delta_k$ 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* $\langle S, \mathsf{Act}, \to \rangle$*, where* $S$ *is a set of states,* $\mathsf{Act}$ *is a set of actions, and* $\to \subseteq S \times \mathsf{Act} \times Dist(S)$ *is the transition relation.*

We often write $s \xrightarrow{\alpha} \Delta$ for $(s, \alpha, \Delta) \in \to$, and $s \xrightarrow{\alpha}$ for $\exists \Delta : s \xrightarrow{\alpha} \Delta$. In a pLTS actions are only performed by states, in that actions are given by relations from states to distributions. But in general we allow distributions over states to perform an action. For this purpose, we *lift* these relations so that they also apply to distributions [5].

▶ **Definition 2.** *Let* $\mathcal{R} \subseteq S \times Dist(S)$ *be a relation from states to distributions in a pLTS. Then* $\mathcal{R}^\circ \subseteq Dist(S) \times Dist(S)$ *is the smallest relation that satisfies the two rules: (i)* $s \;\mathcal{R}\; \Theta$ *implies* $\overline{s} \;\mathcal{R}^\circ\; \Theta$*; (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 countable index set.*

We apply this operation to the relations $\xrightarrow{\alpha}$ in the pLTS for $\alpha \in \mathsf{Act}_\tau$, where we also write $\xrightarrow{\alpha}$ for $(\xrightarrow{\alpha})^\circ$. Thus as source of a relation $\xrightarrow{\alpha}$ we now also allow distributions. But note that $\overline{s} \xrightarrow{\alpha} \Delta$ is more general than $s \xrightarrow{\alpha} \Delta$ because if $\overline{s} \xrightarrow{\alpha} \Delta$ then there is a collection of distributions $\Delta_i$ and probabilities $p_i$ such that $s \xrightarrow{\alpha} \Delta_i$ for each $i \in I$ and $\Delta = \sum_{i\in I} p_i \cdot \Delta_i$ with $\sum_{i\in I} p_i = 1$.

Let $\mathcal{R} \subseteq S \times S$ be a relation between states. It induces a special relation $\hat{\mathcal{R}} \subseteq S \times Dist(S)$ between states and distributions by letting $\hat{\mathcal{R}} \stackrel{def}{=} \{(s, \overline{t}) \mid s \;\mathcal{R}\; t\}$. Then we can use Definition 2 to lift $\hat{\mathcal{R}}$ to be a relation $(\hat{\mathcal{R}})^\circ$ between distributions. For simplicity, we combine the above two lifting operations and directly write $\mathcal{R}^\circ$ for $(\hat{\mathcal{R}})^\circ$ in the sequel, with the intention that a relation between states can be lifted to a relation between distributions via a special application of Definition 2. In this particular case, it holds that $\Delta \;\mathcal{R}^\circ\; \Theta$ implies $\Theta \;(\mathcal{R}^{-1})^\circ\; \Delta$, where $s \;\mathcal{R}\; t$ iff $t \;\mathcal{R}^{-1}\; s$ for any $s, t \in S$. This way of lifting relations has elegant mathematical characterisations; see [6] for more details.

## 3 Quantum CCS

We introduce a quantum extension of classical CCS (qCCS) which was originally studied in [8, 14, 10]. 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,

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

**Figure 1** Free quantum variables

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:

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

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 [13]. 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.

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

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 $\rho$ 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 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.

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

We use $cn(\alpha)$ for the set of channel names in action $\alpha$. 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 $||$ 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** ([7]). *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 $\Theta$ 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. Two configurations are* ground bisimilar *if they are related by some ground bisimulation.*

## 4 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. 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 initializes 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 initialization. The algorithm keeps updating two sets: the above mentioned $W$ and the set $N$ for non-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

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

**Algorithm 1** Bisim(t,u)

**Require:** A pair of initial states for matching $t$,$u$.

**Ensure:** A boolean value $\theta$ showing if two pLTSs are bisimilar and a set of non-bisimilar state pairs $N$.

1: **function Bisim**$(t, u)$
2: **return Match**$(t,u,W)$
3:
4: **function Match**$(t, u, W)$ $\triangleright$ $t = \langle t, \rho\rangle$ *and* $u = \langle u, \sigma\rangle$
5: **if** $t, u \in W$ **then**
6: $\theta := \mathtt{tt}$
7: **else**
8: **for** $\gamma \in \mathtt{Act}(t, u)$ **do**
9: $(\theta_\gamma, N_\gamma)$:=**MatchAction**$(\gamma,t,u,W)$
10: $\theta$:=$\bigwedge_\gamma \theta_\gamma \wedge qv(t) = qv(u) \wedge tr_{\overline{qv(t)}}(\rho) = tr_{\overline{qv(t)}}(\sigma)$
11: $N = \bigcup_\gamma N_\gamma$
12: **if** $\theta = \mathtt{ff}$ **then** $N$:=$N \cup \{(t, u)\}$
13: **return** $(\theta, N)$
14:
15: **function MatchAction**$(\gamma, t, u, W)$
16: **switch** $\gamma$ **do**
17: **case** $c!$
18: **for** $t \xrightarrow{c!e_i} t_i$ **do**
19: **for** $u \xrightarrow{c!e'_j} u_j$ **do**
20: $(\theta_{ij}, N_{ij}) :=$ **Match**$(t_i, u_j, W \cup \{(t, u)\}$
21: **return** $(\bigwedge_i(\bigvee_j(\theta_{ij} \wedge e_i = e'_j)) \wedge \bigwedge_j(\bigvee_i(\theta_{ij} \wedge e_i = e'_j)), \bigcup_{ij} N_{ij})$
22: **case** $\tau$
23: **for** $t \xrightarrow{\tau} \Delta_i$ **do**
24: **for** $u \xrightarrow{\tau} \Theta_j$ **do**
25: $(\theta_{ij}, N_{ij}) :=$ **MatchDistribution**$(\Delta_i, \Theta_j, W \cup \{(t, u)\}$
26: **return** $(\bigwedge_i(\bigvee_j \theta_{ij}) \wedge \bigwedge_j(\bigvee_i \theta_{ij}), \bigcup_{ij} N_{ij})$
27: **otherwise**
28: **for** $t \xrightarrow{\gamma} t_i$ **do**
29: **for** $u \xrightarrow{\gamma} u_j$ **do**
30: $(\theta_{ij}, N_{ij}) :=$ **Match**$(t_i, u_j, W \cup \{(t, u)\}$
31: **return** $(\bigwedge_i(\bigvee_j \theta_{ij}) \wedge \bigwedge_j(\bigvee_i \theta_{ij}), \bigcup_{ij} N_{ij})$
32:
33: **function MatchDistribution**$(\Delta, \Theta, W)$
34: **for** $t_i \in \lceil\Delta\rceil$ and $u_j \in \lceil\Theta\rceil$ **do**
35: $(\theta_{ij}, N_{ij}) :=$ **Match**$(t_i, u_j, W)$
36: $R$:=$\{(t_i, u_j) | (t_i, u_j) \notin \bigcup_{ij} N_{ij}\}^*$
37: **return** (**Check**$(\Delta,\Theta,R) \wedge \bigvee_{ij} \theta_{ij}, \bigcup_{ij} N_{ij})$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**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 $\tilde{q}$.

are three possibilities that two states are deemed be non-bisimilar: (1) one state has a transition that the other cannot match, (2) they do have 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 places, we add the state pair into $N$.

An auxiliary function $\textbf{Act}(t, u)$ 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 $\textbf{MatchAction}(\gamma, t, u, W)$. This function discovers next pairs or 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 $\textbf{MatchDistribution}(\Delta, \Theta, R)$ 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 $\textbf{Check}(\Delta, \Theta, R)$ is used for checking the lifting condition of the bisimulation relation. Besides the lifting condition, we

check the disjunction of the returning boolean values from function **Match**.

▶ **Definition 4** (Lifting Condition). *Let $R \subseteq Dist(Con) \times Dist(Con)$ be the (strong) open bisimulation relation between two distributions, then for any $\mu, \nu \in Dist(Con)$, $\mu R \nu$ can imply that:*

*(1) The relation satisfies the lifting condition, that is $\mu = \sum_{i \in I} p_i C_i$, for each $i \in I$, $C_i R D_i$ for some $D_i$, and $\nu = \sum_{i \in I} p_i D_i$.*
*(2) The set $I$ is not an empty set, s.t. $\exists C, D \in Con, \mu(C) > 0 \wedge \nu(D) > 0$.*

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

▶ **Theorem 5** (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 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 exists actions enabled in one pLTS but not in the other, then the two pLTSs are not bisimilar and then the whole algorithm terminates. ◀

▶ **Theorem 6** (Correctness). *Given two states $t$ and $u$ from two pLTSs, **Bisim**$(t, u)$ returns true if and only if they are bisimilar.*

**Proof.** The proof of the correctness is similar to that in [12]. However, we do not consider the *most general boolean* constraints each bisimilarities. On the other hand, we have an additional procedure **MatchDistribution** between **Match** and **MatchAction** in some cases to compute the constraint that relates two probability distributions. The details of the proof are presented in Appendix B.

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

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

▶ **Theorem 7** (Complexity). *Let the number of nodes in two transition graphs reachable from $t$ and $u$ is $n$. The time complexity of function **Bisim**$(t, u)$ is $O(n^5/log\ n)$ and the space complexity of it is $O(n^2)$.*

**Proof.** The number of state pairs is bounded by $n^2$. Since the graphs there are finite trees, for each pair of states, the number of comparisons of transitions could be $n^2$ in the worst case. So the visited state pairs set $W$ contains at most $O(n^2)$ elements.

And if in each call of **MatchAction**, it call **MactchDistribution** other than **Match**, there will cost more time checking the condition of lifting operation through **Check**. In previous work [6], we know **Check** cost time $O(n^3/log\ n)$. As a result, the execution of **Bisim**$(t, u)$ takes at most $O(n^5/log\ n)$ time in total. ◀

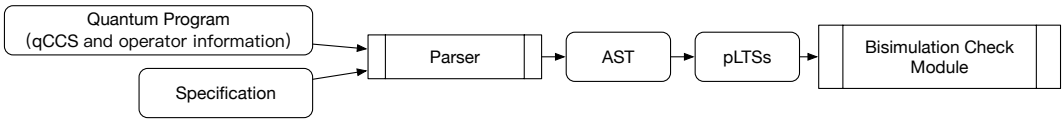


**Figure 3** Verification workflow.

# 5 Experimental Results

In this section, we report an implementation of our approach and provide several classical quantum communication protocols in qCCS as the use cases for our experiment. We show that our approach is able to distinguish the bisimilarity, and several improvements can be made according to the result of the algorithm.

## 5.1 Implementation

Our tool is implemented in Python 3.7. Its workflow is illustrated in Fig. 3. The input of this tool is a quantum program and its specification described in qCCS. The parser for the qCCS program is also implemented by us using Python. Execution of the tool yields a terminal output showing the details of the whole process, including the pLTS generation and the checking algorithm, and the result of the checking by a table mapping each pair of pLTS states to its most general boolean. The tool invokes Z3 solver to verify the most general boolean of the initial state pair. A counterexample will be given if the boolean can be unsatisfied.

**pLTS Generation.** The tool inputs programs codes containing three parts,a description of process behaviors, an initialization of the variables and a set of user-defined quantum gates. Process behaviors are described using the notation of qCCS syntax, separated by semicolons. Quantum gates can be defined through a set of Kraus operators, they are also separated by semicolons. The intermediate output of the module is the pLTS which will be used as the input of bisimilarity checking module.

**Bisimulation Checking.** We implement the previously defined ground bisimlarity checking algorithm to verify the generated pLTSs. The input needs two pLTSs, one for protocol description and another for specification description. They are processed by the pLTS generation module. We start at the initial states of these two pLTSs. The result of the module is also the final result of the tool presenting whether these two pLTSs are bisimilar, always with a set preserving non-bisimilar state pairs.

## 5.2 Examples: Quantum Communication Protocols

**Super-dense Coding Protocol.** There are two roles *Alice* and *Bob*. To simplify the experiment, we only consider the smallest case of the protocol, sending only one qubit. So there is totally one entanglement on two qubits in this example. Besides the Clifford operators, we use a quantum operation $Set^{\Psi}$ to present the generation of EPR state instead of using a combination of the quantum gates. The operation elements of $Set^{\Psi}$ is $\{|\beta_{00}\rangle\langle 00|, |\beta_{00}\rangle\langle 01|, |\beta_{00}\rangle\langle 10|, |\beta_{00}\rangle\langle 11|\}$. The measurement is according to the computational basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. The specification of the super-dense protocol is defined as *Bob* sets the 2-qubit variable to the value according to the classical value he received from *Alice*.

**Quantum Teleportation Protocol.** In this example, there are still two roles. The operators we used here is similar with the last example containing Clifford operators, $Set^{\Psi}$ and the measurement according to the computational basis. However, we need one more entanglement and one more qubit even if we just consider the smallest case. As there are more than one entanglements between these qubits, although the measurement is just applied on a part of them, it may also affect the rest qubits. We consider the final result of this protocol is presented on the third qubit, $Bob's$ qubit. It should become the same value of the first qubit. So the specification of that can be presented by applying a $SWAP$ operation between the first and the third qubit.

**Quantum Secret Sharing.** In this example, there are three roles. The operators used here are almost the operators we have used in teleportation protocol except that we use a quantum operation $Set^{GHZ}$ to present the generation of 3-qubit maximally entangled states. This example is closed to the teleportation example, the difference is that it makes four qubits entangled together. So its specification becomes applying a $SWAP$ operation between the first and the fourth qubit.

**BB84 Quantum Key Distribution Protocol.** In BB84 protocol, there is no entanglement at all, its method needs generating qubits on different basis and using different measurement method without any contacts in advance with the other side. If someone tries intercepting the information, the qubits might be measured in wrong basis, it brings a possibility that *Alice* and *Bob* can be aware of the attack. So the protocol uses one more kind of measurement which is according to the diagonal basis $\{|+\rangle, |-\rangle\}$. In common use case, BB84 will send a sequence of the qubits while qubits will not influence each other. We consider two kinds of result of the communication. First case is that *Alice* and *Bob* choose the same measurement then the results they get are also the same. Another case is that they choose different measurements then the result is discarded at this time. In the specification, we get results from the same sequence instead of two result sequence separately. Considering results from both sides is always the same, this operation will not bring any difference.

**BB84 Protocol with an Eavesdropper.** This example is an extension of the BB84 example, supposing there is an eavesdropper attending into the communication. There have three roles and the new role *Eve* also randomly choose the measurement just as what *Alice* and *Bob* do. The specification is also similar with the one without an eavesdropper. It is possible that the eavesdropper will be recognized. It is a new result of the program. We conclude these results into three messages: emitting through the channel *alarm* as the measurement methods are not matched; emitting through the channel $fail$ as the measurement methods are matched while the eavesdropping is recognized; normal emitting as the communication finished without recognizing the eavesdropper.

## 5.3 Experimental Results

We conducted experiments on those quantum communication protocols, and improved our input program according to the experiment results. The results were obtained on a macOS machine with an Intel Core i7 2.5 GHz processor and 16GB of RAM.

**Experimental Results and Improvement.** Table 1 provides a summary of our experimental results over those four examples. In each case, we report the bisimilarity, the number of non-bisimilar states pair in $N$ and the runtime of our checking algorithm.

We verify the super-dense coding with two different initial valuations of variable $x$ in the first two lines. In the case $x = 1$, we can check that protocol and its specification are bisimilar. However, in the case $x = 5$, when none of the four branches is chosen, they are not bisimilar because of the different length of the trace. The result shows that the program misses the solution for the valuation out of the expected scale. We improve the program through adding a new branch solving all the unexpected value. The result of the improved program is presented on the third line.

We also verify the teleportation with several different initial valuations of qubits. The input qubits are used to generate the EPR state or other fixed value in other examples, what is different from this example. The qubit $q_1$ in the inputs decides the final state of the qubit $q_3$. We can find that there is no difference between the result information from different inputs.

Another example brings a non-bisimilarity is on the second line from the bottom, which is the BB84 protocol considering the eavesdropper. *Alice* and *Bob* will make an alert if their measurement methods are not matched. The parallelism between the final test process and them leads to the process continues behaving some actions. That is not what the specification exactly describes. To improve this program, we modify the behavior, move the alert to the test process. *Alice* and *Bob* only send messages when they find they use different measurements. As a result, on the last line of the table, we find the program is bisimilar with the specification.

**Discussion.** Not all the cases of Table 1 present the size of the non-bisimilar states set $N$, as the checking algorithm has terminated at an early point. Furthermore, to ensure the bisimilarity between program with a large set of states and its specification requires much more time, over 24 times of the runtime of checking non-bisimilarity. However, the runtime of finding two pLTSs are non-bisimilar is not that long enables us to try making improvement in an acceptable time waiting feedback.

## 6 Conclusion and Future Works

In this paper we have presented an on-the-fly algorithm verifying strong ground bisimulation for quantum programs in qCCS. And then we have developed a tool for bisimilarity checking basing on the algorithm. The input is encoded with the notation of qCCS, which enables us to translate the program to corresponding finite pLTS depending on its operational semantics. We proved our algorithm terminates on checking two finite tree-structural pLTSs and figures out the bisimilarity. To show its performance, we further made experiments on several quantum communication protocols such as BB84. The experimental results have showed the algorithm is able to provide hints for two kinds of improvement: (1) supplementing the missing cases of the programs; (2) adjusting how the parallel behaviours collaborate with each other.

There are still many questions remaining for further study.Firstly, the bisimulation checking may not only several possible inputs which we can enumerate all of them in a short time. One of the solution of that is to introduce the idea of symbolic bisimulation proposed in [9]. Symbolic bisimulation uses an accumulation of the super-operators instead of a density operator to present the state which allows us to verify the programs with arbitrary inputs. However, the normalizing operation is also unavailable without the density operator, so it becomes a challenge.

Secondly, it is eye-catching that there are too many invisible action $\tau$ contained in the

| Program | Variables | Bisim | MN | SN | $N$ | Sec |
|---|---|---|---|---|---|---|
| Super-dense coding 1 | $q_1 = \|0\rangle$<br>$q_2 = \|0\rangle$<br>$x = 1$ | Yes | 15 | 15 | 0 | 2.2 |
| Super-dense coding 2 | $q_1 = \|0\rangle$<br>$q_2 = \|0\rangle$<br>$x = 5$ | No | 5 | 12 | - | 2.2 |
| Super-dense coding (modified) | $q_1 = \|0\rangle$<br>$q_2 = \|0\rangle$<br>$x = 5$ | Yes | 15 | 15 | 0 | 2.5 |
| Teleportation 1 | $q_1 = \|1\rangle$<br>$q_2 = \|0\rangle$<br>$q_3 = \|0\rangle$ | Yes | 33 | 15 | 0 | 2.2 |
| Teleportation 2 | $q_1 = \frac{1}{\sqrt{2}}\|0\rangle + \frac{1}{\sqrt{2}}\|1\rangle$<br>$q_2 = \|0\rangle$<br>$q_3 = \|0\rangle$ | Yes | 33 | 15 | 0 | 2.2 |
| Teleportation 3 | $q_1 = \frac{\sqrt{3}}{2}\|0\rangle + \frac{1}{2}\|1\rangle$<br>$q_2 = \|0\rangle$<br>$q_3 = \|0\rangle$ | Yes | 33 | 15 | 0 | 2.2 |
| Secret Sharing 1 | $q_1 = \|1\rangle$<br>$q_2 = \|0\rangle$<br>$q_3 = \|0\rangle$<br>$q_4 = \|0\rangle$ | Yes | 102 | 26 | 0 | 2.6 |
| Secret Sharing 2 | $q_1 = \frac{1}{\sqrt{2}}\|0\rangle + \frac{1}{\sqrt{2}}\|1\rangle$<br>$q_2 = \|0\rangle$<br>$q_3 = \|0\rangle$<br>$q_4 = \|0\rangle$ | Yes | 102 | 26 | 0 | 2.5 |
| Secret Sharing 3 | $q_1 = \frac{\sqrt{3}}{2}\|0\rangle + \frac{1}{2}\|1\rangle$<br>$q_2 = \|0\rangle$<br>$q_3 = \|0\rangle$<br>$q_4 = \|0\rangle$ | Yes | 102 | 26 | 0 | 2.5 |
| BB84 | $q1 = \|0\rangle$<br>$q2 = \|0\rangle$ | Yes | 151 | 131 | 304 | 4.7 |
| BB84 (with eavesdropper) | $q1 = \|0\rangle$<br>$q2 = \|0\rangle$<br>$q3 = \|0\rangle$ | No | 1243 | 763 | - | 74.6 |
| BB84 (with eavesdropper & modified) | $q1 = \|0\rangle$<br>$q2 = \|0\rangle$<br>$q3 = \|0\rangle$ | Yes | 1179 | 779 | 17272 | 1834 |

**Table 1** Experimental Results. The columns headed by **MN** and **SN** show the number of nodes contained in the generated pLTSs of implementation and specification. Column $N$ shows the size of the non-bisimilar pairs set $N$. Column **SEC** shows verification runtimes of the tool, times are in seconds.

programs especially the specification programs matching the internal communications and quantum operations in the implementation programs. To deal with this problem, we are going to implement a weak bisimulation checking algorithm wipe out those invisible actions.

## References


1. 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. URL: `https://dl.acm.org/citation.cfm?id=3231597`.
2. 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.
3. 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.
4. 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.
5. 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.
6. Yuxin Deng and Wenjie Du. Logical, metric, and algorithmic characterisations of probabilistic bisimulation. Technical Report CMU-CS-11-110, Carnegie Mellon University, March 2011.
7. 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.
8. Y. Feng, R. Duan, Z. Ji, and M. Ying. Probabilistic bisimulations for quantum processes. *Information and Computation*, 205(11):1608–1639, 2007.
9. Yuan Feng, Yuxin Deng, and Mingsheng Ying. Symbolic bisimulation for quantum processes. *ACM Transactions on Computational Logic*, 15(2):1–32, 2014.
10. Yuan Feng, Runyao Duan, and Mingsheng Ying. Bisimulation for quantum processes. In *Proc. POPL'11*, pages 523–534. ACM, 2011.
11. Yuan Feng, Nengkun Yu, and Mingsheng Ying. Model checking quantum markov chains. *J. Comput. Syst. Sci.*, 79(7):1181–1198, 2013. URL: `https://doi.org/10.1016/j.jcss.2013.04.002`, `doi:10.1016/j.jcss.2013.04.002`.
12. Matthew Hennessy and Huimin Lin. Symbolic bisimulations. *Theor. Comput. Sci.*, 138(2):353–389, 1995. URL: `https://doi.org/10.1016/0304-3975(94)00172-F`, `doi:10.1016/0304-3975(94)00172-F`.
13. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
14. M. Ying, Y. Feng, R. Duan, and Z. Ji. An algebra of quantum processes. *ACM Transactions on Computational Logic*, 10(3):1–36, 2009.


# A Examples in qCCS

## A.1 Super-dense Coding Protocol

Super-dense coding is proposed by Bennett and Wiesner in 1992 [4]. 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 bases 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 an one-qubit operation, and these EPR states are mutually orthogonal. 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 $\sigma^x$ 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. Considering $x$ could be presented in a two-bit string, the protocol exactly transmits two classical bits of information by sending one qubit from *Alice* to *Bob*.

**Implementation.** Now we design the program of super-dense coding protocol in qCCS as follows:

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

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

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

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

where $\mathcal{CN}$ is the controlled-not operation and $\mathcal{H}$ is the Hadamard operation, $Set^{\Psi}$ 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 $\sigma^i$ 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 super-dense coding 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 transforming the current state into the state decided by the value of $i$ like $Set^{\Psi}$.

**Improved Super-dense Coding Protocol.** We improve the program through adding an extra solution for the value $i \neq 1,2,3,4$. We send a message alarming we have encountered such case and skip all the rest operations. The new program of $Sdc$ is:

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

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

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

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

And we adjust the specification as the program has a new branch, so it is defined as:

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

## A.2 Quantum Teleportation Protocol

Quantum teleportation [3] 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 communications channel. It makes the use of the maximally entangled states that the post-measurement state can be known from the result of partial measurement for a set of entangled states. Let sender and receiver to be *Alice* and *Bob* as defined in super-dense coding example, 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 $\mathcal{CN}$ operation on $q_1$ and $q_2$ followed by a $\mathcal{H}$ operation on $q_1$.
**(3)** *Alice* measures $q_1$ and $q_2$ and sends the outcome $x$ to *Bob*.
**(4)** *Bob* applies corresponding $\sigma^x$ 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.** The program of quantum teleportation protocol can be encoded in qCCS as follows:

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

$$Bob \stackrel{def}{=} \underline{c}_B?q_3.e?x.\sum_{0\leq i\leq 3}(\textbf{if } x=i \textbf{ then } \sigma^{i}[q3].\textbf{nil});$$

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

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

where the operators used are all already declared before.

**Specification.** The specification of quantum teleportation protocol can also be described in qCCS. To show the soundness of $Tel$, it suffices to prove that $Tel$ is bisimilar to an swap operation between the first and the thrid qubits, that is $\mathcal{SWAP}_{1,3}[q_1,q_3]$. The program can be coded as follow:

$$Spec \stackrel{def}{=} \tau^{13}.\mathcal{SWAP}[q_1,q_3].\textbf{nil}.$$

## A.3 BB84 Quantum Key Distribution Protocol

BB84 is the first quantum key distribution protocol developed by Bennett and Brassard in 1984 [2]. It 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 the 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 sender and receiver to be *Alice* and *Bob* as defined in previous examples, 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 $i$th bits of $\tilde{q}$ is $|x_y\rangle$ where $x$ and $y$ are the $i$th 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 $i$th qubit of $\tilde{q}$ he received from *Alice* according to the basis determined by the $i$th 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 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 eavesdropper, let $\tilde{K'}_a$ and $\tilde{K'}_b$ to be the remaining bits of $\tilde{K}_a$ and $\tilde{K}_b$ with size $k$, *Eve*, *Alice* and *Bob* 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.

**(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 substring 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.** The program we written here only contains one qubit instead of a sequence of qubits, however, it is enough to reflect all the cases could occur. The other qubits used here are auxiliary qubits for *Ran* operation.

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

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

$$BB84 \stackrel{def}{=} (Alice||Bob) \setminus \{a2b, b2a, \underline{A2B}\}$$

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 $\epsilon$, meaning it is empty, if they do not match.

**Specification.** Its specification can be defined as follow using the same operations:

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

**Implementation with an Eavesdropper.** Then we proceed to describe the protocol with an eavesdropper. We extend the processes *Alice* and *Bob* with a process for eavesdropper detection.

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

$$Bob' \stackrel{def}{=} key_b?K'_b.a2b?x.a2b?K''_a.b2a!SubStr(K'_b,x). \\ (\mathbf{if}\ SubStr(K'_b,x)=K''_a\ \mathbf{then}\ key'_b!RemStr(K'_b,x).\mathbf{nil} \\ \mathbf{else}\ alarm_b!0.\mathbf{nil})$$

where there are three more special operations:

- $Pstr$ is a measurement which is similar to $Ran$, 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 $SubStr(K,x)$.

After that, we give the definition of the eavesdropper:

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

With the attending 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 $f_b(\underline{A2B}) = \underline{E2B}$.

We use a test process to conclude the final result:

$$Test \stackrel{def}{=} key'_a?x.key'_b?y.key'_e?z. \\ (\mathbf{if}\ x\neq y\ \mathbf{then}\ fail!0.\mathbf{nil} \\ +\ \mathbf{if}\ x=y\ \mathbf{then}\ key_e!z.skey!x.\mathbf{nil});$$

$$BB84 \stackrel{def}{=} (Alice||Bob||Alice'||Bob'||Eve||Test)\setminus C$$

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

**Specification.** The specification of that can be defined as:

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

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 randomly generate the value of $v$ if $x$ and $y$ do not match and give $v$ the value of $z$ if they match.

**Improved BB84 Protocol with an Eavesdropper.** As the reason why the problem occurred is that the operation $alarm$ is not concluded in $Test$ and the implementation have more behaviours than what the specification requires. So we only refine the program of implementation, adding a message communication with $Test$:

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

## B Proving Correctness of the Ground Algorithm

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

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

▶ **Lemma 8.** *If $N_1 \cap N_2 = \emptyset$ then $\mathtt{R}(t, u, W, N_1)$ and $\mathtt{R}(t, u, W, N_2)$ implies $\mathtt{R}(t, u, W, N_1 \cup N_2)$.*

**Proof.** Straightforward from the definition of $\mathtt{R}$. ◀

We define the verification conditions of our three matching functions.

▶ **Definition 9.** ***Match***$(t, u, W)$ *is true if the following conditions ar satisfied:*

- *(C1) $W \cap N = \emptyset$ and*
  - *if $(t, u) \in W$, then $(t, u) \notin N$,*

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

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

▶ **Definition 10.** $\boldsymbol{MatchAction}(\gamma,t,u,W)$ *is true if all the following conditions are satisfied:*

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

▶ **Definition 11.** $\boldsymbol{MatchDistribution}(\Delta,\Theta,W)$ *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* $\exists (t_i,u_j) \notin N$.
- *(D2) Let* $t \xrightarrow{\alpha} \Delta, u \xrightarrow{\alpha'} \Theta$, $\mathtt{R}(t,u,W,N)$.

▶ **Proposition 12.** *Let* $\boldsymbol{MatchAction}_\gamma(\gamma,t,u,W)$ *is the execution of* $\boldsymbol{MatchAction}$ *with action* $\gamma$*. If* $\boldsymbol{MatchAction}_\gamma(\gamma,t,u,W)$ *is true for each action* $\gamma$ *then* $\boldsymbol{Match}(t,u,W)$ *is also true, where 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 10, we have $W \cap N = \emptyset$. Since the verified pLTS is a finite tree, if they reach the leaf states of the pLTSs, there should have $\theta = true$ and $N = \emptyset$, at the same time it satisfies that $(t,u) \notin W \wedge (t,u) \notin N$. Furthermore, we have $t \sim u$ in such case. According to the structure of the function, $(t,u)$ will be added in to $N$ if $\theta$ is false. Overall, $C1$ is satisfied.

From condition (M2) and (M3), $\mathtt{R}(t,u,W,N_\gamma)$ exists. According to Lemma 8, we have the condition that $\mathtt{R}(t,u,W,\bigcup_\gamma N_\gamma)$, it satisfies $C2$. ◀

▶ **Proposition 13.** *Suppose* $(t,u) \notin W$*. If* $\boldsymbol{Match}(t_i,u_j,W \cup \{(t,u)\})$ *is true for all actions* $\gamma \neq \tau$ *where exists transitions* $(t \xrightarrow{\gamma} t_i,\ u \xrightarrow{\gamma} u_j)$ *or* $\boldsymbol{MatchDistribution}(\Delta_i,\Theta_j,W \cup \{(t,u)\})$ *is true for all actions* $\gamma = \tau$ *where exists transitions* $(t \xrightarrow{\tau} \Delta_i, u \xrightarrow{\tau} \Theta_j)$ *then* $\boldsymbol{MatchAction}(\gamma,t,u,W \cup \{(t,u)\})$ *is true where* $\theta = \bigwedge_i(\bigvee_j \theta_{ij}) \wedge \bigwedge_j(\bigvee_i \theta_{ij})$, $N = \bigcup N_{ij}$.

**Proof.** From the structure of **MatchAction**, $(t,u)$ does not exist in $W$, and $(t,u)$ can not be added into $N$ 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 leafs 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 $(t_i,u_j) \notin N_{ij}$ and $N_{ij} = \emptyset$. So there is $t_i \sim u_j$.

If it is not the leaf node, by $(C2)$, we have $\mathtt{R}(t_i,u_j,\{(t,u)\} \cup W, N_{ij})$. Since $N = \bigcup_{ij} N_{ij}$, we get if actions match $(t_i'',u_j'') \notin W \wedge (t_i'',u_j'') \notin N \implies t_i'' \sim u_j''$. By definition of the bisimulation, $(M3)$ is satisfied. So we also get if actions match $(t_i',u_j') \notin W \wedge (t_i',u_j') \notin N \implies t_i' \sim u_j'$. If $\theta$ is true, as $\theta = \bigwedge_i(\bigvee_j \theta_{ij}) \wedge \bigwedge_j(\bigvee_i \theta_{ij})$, so there exists $\theta_{ij}$ which is $true$, then there is $(t_i,u_j) \notin N_{ij}$. Similarly, by definition of the bisimulation, $(M2)$ is also satisfied.

The final case we need consider is the distribution $(\Delta,\Theta)$ instead of a node. If $\theta$ is true, then $\theta_{ij}$ returned from **Check** should also be $true$. So there must exist **Match** returns $true$ support the relation that $(t_i,u_j) \notin N \implies t_i \sim u_j$. ◀

▶ **Proposition 14.** *Suppose* $\forall t_i \in \lceil\Delta\rceil, u_j \in \lceil\Theta\rceil, (t_i, u_j) \notin W$. *If* $\boldsymbol{Match}(t_i, u_j, W)$ *is true then* $\boldsymbol{MatchDistribution}(\Delta, \Theta, W)$ *is true where* $\Delta$ *and* $\Theta$ *satisfy the condition for lifting condition,* $\theta = \boldsymbol{Check}(\Delta, \Theta, R) \wedge \bigvee_{ij} \theta_{ij}$ *and* $N = \bigcup_{ij} N_{ij}$.

**Proof.** According to the verification conditions of **March**, as all the **Match**$(t_i, u_j, W)$ have been finished before we get $R$ and call **Check**. If $\Delta \sim \Theta$, then we have $(t_i, u_j) \notin N \implies t_i \sim u_j$. ◀