Verifying Strong Ground Bisimilarity of Quantum Communication Protocols

First Author[1], Second Author[2,3], and Third Author[3]

[1] University, Princeton NJ 08544, USA
[2] Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany
lncs@springer.com
http://www.springer.com/gp/computer-science/lncs
[3] ABC Institute, Rupert-Karls-University Heidelberg, Heidelberg, Germany
{abc,lncs}@uni-heidelberg.de

Abstract. This paper gives a strong ground bisimulation verification algorithm for quantum programs. We further implement the algorithm that enable us to make experiments on existing quantum communication protocols. As a preparation of the experiments, we encode the quantum communication protocol into a quantum program. Then we check whether a quantum program is bisimilar with its specification. According to the results, we make some improvements on the quantum programs.

Keywords: Quantum programs · Verification · Bisimualtion.

1 Introduction

2 Preliminaries

3 qCCS

4 Bisimulation Verification

In this section, we give an algorithm to verify the strong ground bisimulation between two pLTSs and show its implementation.

4.1 A Strong Ground Bisimilarity Checking Algorithm

The algorithm is based on the work of [**?**]. The main function is $\mathbf{Bisim}(t, u)$, its job is to initialize the start states pair (t, u), visited states pair W which is an empty set and then find the bisimulation basing on that initialization. The difference between it and the previous work in several aspects.

The algorithm keeps updating two sets W for visited states pairs and N for non-bisimilar states pairs. The function $\mathbf{Match}(t, u, W)$ invokes a depth-first traversal to match a pair of states (t, u) with all their possible behaviors. The states pair is checked to be non-bisimilar if one of their transitions are

not matched or their quantum variables, so do the quantum registers, are not matched. Then the algorithm adds the current states pair into N.

An auxiliary function $\mathbf{Act}(t, u)$ is called in **Match** to discover the next action that both two states can behave. If both two states have no more action it returns an empty set. Furthermore, if there is only one of them has no more action it will lead to a non-bisimilarity. It makes the algorithm more efficient as it terminates at an early time point if those pLTSs are indeed not bisimilar. Next we prove why we can ensure that.

Theorem 1 (Early termination). *If the algorithm reaches a leaf state of the tree-like pLTS while the state of the other pLTS is not leaf state, then these two pLTSs are not bisimilar.*

Proof. We consider it on the aspect of the length of the traces. From the structure of the algorithm, each time **MatchAction** is called **Act** will be called before it. So the we can ensure that two states have the same action to behave. There exists the trace that

$$T = \langle t_0, \rho\rangle \xrightarrow{\gamma_0} ... \xrightarrow{\gamma_i} \langle t_i, \rho\rangle \text{ and } U = \langle u_0, \sigma\rangle \xrightarrow{\gamma_0} ... \xrightarrow{\gamma_i} \langle u_i, \sigma\rangle.$$

Let $|T| = |U| = n$, if one of these states u_i is not leaf state, then there has a longer trace $|U'| = n + 1$. As there is no loop contained, the trace $|T'| = n + 1$ does not exist, so we can not find a trace has the same length as the other one. According to the definition of the open bisimulation, these two pLTS should not satisfy the relation.

The other set W is update in function $\mathbf{MatchAction}(\gamma, t, u, W)$. It discovers next states pairs according to the action γ and recursively invokes the function **Match** when there is a pair of states or **MatchDistribution** when there is a pair of distributions of states. The current states pair is added to W when it invokes new function.

The $\mathbf{MatchDistribution}(\Delta, \Theta, R)$ is an extra step if we match a pair of distributions of states instead of a single pair of states. It returns a boolean value indicating if the distributions are bisimilar. It continues invoking **Match** to match next pair of states from the pair of states distributions. After checking the bisimilarity of their states, the function generates an equivalence relation of the states from the distribution those who are not contained in the non-bisimilar states set N. Another auxiliary set $\mathbf{Check}(\Delta, \Theta, R)$ is used for checking the lifting condition of the bisimulation relation. Besides the lifting condition, we check the disjunction of the returning boolean value from **Match** functions. The function return such result basing on the following theorem.

Theorem 2 (Bisimulation of distributions).

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. $\exists C, D \in Con, \mu(C) > 0 \wedge \nu(D) > 0$.

Proof. From the definition of the lift operation, the condition must be satisfied. And we need to filter the case that two distributions have no behaviour in common, so there need at least a pair of states is bisimilar.

The correctness of the algorithm is presented in the theorem below.

Theorem 3 (Termination). *For two states t and u, the algorithm always terminate.*

Proof. So far there is no while-loop in the qCCS, that brings convenience to the proof of termination. Starting at the initial pair of states, the next action to do will be detected in the 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 states 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 to do in one of the pLTS while another one does not, that means they are not strong bisimilar and then the whole algorithm terminates.

4.2 Implementation

Our tool is implemented in Python 3.7. Its workflow is illustrated in Fig. **??**. The input is a quantum program and its specification, both of them are described in qCCS. 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 their variables and a set of user-defined quantum gates. Process behaviors are described in qCCS semantics. Processes are 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.

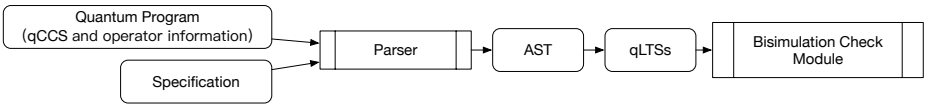

Fig. 1. Verification workflow.

5 Case Studies and Experiments

We provide several classical quantum communication protocols described in qCCS as the use cases for our experiment.

5.1 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 in this example, there is totally one entanglement on two qubits. Besides the Clifford operators, we use a quantum operation Set^{Ψ} to present the generation of Bell state instead of the combination of the quantum gates. The operation elements of Set^{Ψ} 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^{Ψ} and the measurement according to the computational basis. However, we need one more entanglement and one more qubit if we just consider the smallest case. As there are entanglements between these qubits, the measurement on just a part of them may also affect the rest qubits. We considers the final result of this protocol, that is the third one, $Bob's$ qubit, becomes 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.

BB84 Quantum Key Distribution Protocol

BB84 Protocol with an Eavesdropper This example is an extension of the BB84 example, supposing there is an eavesdropper attending into the communication.

5.2 Experiment Results

Program	**Bisimilarity**	**Size of** N	**Runtime**(s)
super-dense coding	No	-	2.6
super-dense(modified)	Yes	0	2.5
Teleportation	Yes	0	2.7
BB84	Yes	304	4.7
BB84(with eavesdropper)	No	-	74.6
BB84(with eavesdropper & modified)	Yes	17272	1834

6 Conclusion and Future Works

References

1. Feng, Y., Deng, Y., Ying, M.: Symbolic Bisimulation for Quantum Processes. ACM Trans. Comput. Log. **15**(2), 14:1–14:32 (2014)
2. Author, F.: Article title. Journal **2**(5), 99–110 (2016)
3. Author, F., Author, S.: Title of a proceedings paper. In: Editor, F., Editor, S. (eds.) CONFERENCE 2016, LNCS, vol. 9999, pp. 1–13. Springer, Heidelberg (2016). https://doi.org/10.10007/1234567890
4. Author, F., Author, S., Author, T.: Book title. 2nd edn. Publisher, Location (1999)
5. Author, A.-B.: Contribution title. In: 9th International Proceedings on Proceedings, pp. 1–2. Publisher, Location (2010)
6. LNCS Homepage, http://www.springer.com/lncs. Last accessed 4 Oct 2017

Algorithm 1 Bisim(t,u)

Require: A pair of initial states for matching t,u.
Ensure: A boolean value θ showing if two pLTSs are bisimilar and a set of non-bisimilar state pairs N.

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

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

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

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