Bisimulation Verifier User Manual

Takahiro Kubota[1], Yoshihiko Kakutani[1], Go Kato[2],
Yasuhito Kawano[2] and Hideki Sakurada[2]

[1] The University of Tokyo, Tokyo, Japan
{takahiro.k11_30,kakutani}@is.s.u-tokyo.ac.jp
[2] NTT Communication Science Laboratories, Kanagawa, Japan
{kato.go,kawano.yasuhito,sakurada.hideki}@lab.ntt.co.jp

Abstract

This is a user manual for the qCCS bisimulation verifier introduced in a paper "Automated Verification of Equivalence on Quantum Cryptographic Protocols". This manual is for the verifier version 0.6 updated on April 29, 2013, which requires ruby's version 1.9.2p290 or 1.8.7.

1 Usage

In this section, the usage of the verifier is introduced.

Usage

Please run `verifier.rb` by ruby.

`$ ruby verifier.rb` *option file*

file is a script for the verifier. The verifier reads *file* and prints the result. If no option is set, the verifier finds two configurations in *file* and verify their bisimilarity using the equations defined in *file*.

Options

`-s`, show all configurations defined in *file* without checking bisimilarity
`-t`, show transition trees, counts the number of transitions and paths for all configurations defined in *file* without checking bisimilarity
`-d`, show information when the verifier returns false
`-u`, show two partial traces when they are not equal
`-v`, show processes, environments and partial traces of two configurations in each step

* If no option is set, only the bisimilarity check procedure runs.
* The options are mutually exclusive.

Examples

```
$ ruby verifier.rb scripts/shor-preskill.scr
$ ruby verifier.rb -s scripts/shor-preskill.scr
$ ruby verifier.rb -v scripts/qtel.scr
$ ruby verifier.rb -d scripts/epr_or_prob.scr
```

2 Scripts

In this section, how to write scripts (*file*) is introduced. Lines that begin from "`# `" are ignored as comments.

2.1 Declaration

Before formalizing processes and environments, symbols need to be declared that are described in processes or environments.

- natural number symbols with the expression `nat` n`;`. A symbol `1` is initially defined in the framework. (example) `nat n; nat m;`
- channel names with the expression `channel` c `:` n`;`, where n is a natural number symbol defined beforehand. Through channel c, quantum variable with length n are communicated. (example) `channel c1 : n;`
- quantum variables with the expression `qvar` q `:` n`;`, where n is the qubit-length of q. (example) `qvar q : n; qvar r : n; qvar s : n;`
- symbols for quantum states (i.e. density operators) with the expression `dsym` X `:` n_1`,`...`,`n_k`;`. X is a quantum state which k quantum variables with qubit-length $n_1, ..., n_k$ are in. (example) `dsym EPR : n,n; dsym PROB : n,n; dsym EVE : m;`
- symbols for superoperators with the expression `operator` $\mathcal{E}$ `:` n_1`,`...`,`n_k`;`. $\mathcal{E}$ acts on quantum variables with qubit-lengths $n_1, ..., n_k$. (example) `operator hadamards : n;`

2.2 Definition of Configurations

Processes, environments, configurations and equations on environments are then defined.

- A process is defined with the expression `process` *process_name* P `end`.
  (example)

```
process hadamards_send
 hadamards[q].c1!q.discard(r)
end
```

- An environment is defined with the expression `environment` *environment_name* ρ `end`.
  (example)

```
environment epr_env
 EPR[q,r] * EVE[s]
end
```

- A configuration is defined with the expression `configuration proc` *process_name* `env` *environment_name* `end`.
  (example)

```
configuration C1
 proc hadamards_send
 env  epr_env
end
```

If more than two configurations are defined, bisimilarity of the first two configurations is verified.

- An equation is defined with the expression `equation` *equation_name* ρ `=` σ `end`, where ρ and σ are environments. For the quantum state, description `__[`$\tilde{q}$`]` is permitted, which matches arbitrary quantum state of $\tilde{q}$.
(example)

```
equation E1
 Tr[q](hadamards[q](EPR[q,r])) = Tr[q](hadamards[q](PROB[q,r]))
end

equation E2
 hadamards[q](hadamards[q](__[q])) = __[q]
end
```

Example

Quantum teleportation Quantum teleportation protocol is formalized as a configuration `Tel` in figure 1. A configuration `TelSpec` is a specification of quantum teleportation, which merely swaps input's and output's quantum states. With equation `E1` and `E2`, `Tel` and `TelSpec` are automatically proven to be bisimilar. Interpretation of constant symbols, quantum operators and quantum states that appear in this example is as follows. Let I, X and Z be identity, bit flip and phase flip operators respectively.

- constant symbol `2` are trivially interpreted to natural number 2. `m` is interpreted to an arbitrary natural number m.
- density matrix symbols `EPR`, `ZERO` and `AFTER` are interpreted to $|00\rangle\langle 00| + |00\rangle\langle 11| + |11\rangle\langle 00| + |11\rangle\langle 11|$, $|00\rangle\langle 00|$ and $|0000\rangle\langle 0000| + |0101\rangle\langle 0101| + |1010\rangle\langle 1010| + |1111\rangle\langle 1111|$, respectively. `ANY` and `EVE` are interpreted to arbitrary quantum states with defined dimension.
- operator symbols `cnot`, `hadamard` and `swap` are trivially interpreted.
- `measure` is interpreted to $\mathcal{E}_{\texttt{measure}}(\rho) = A\rho A^\dagger$, where $A = |00\rangle\langle 00| \otimes I \otimes I + |01\rangle\langle 01| \otimes I \otimes X + |10\rangle\langle 10| \otimes X \otimes I + |11\rangle\langle 11| \otimes X \otimes X$.
- `telproc` is interpreted to $\mathcal{E}_{\texttt{telproc}}(\sigma) = B\sigma B^\dagger$, where $B = |00\rangle\langle 00| \otimes I + |01\rangle\langle 01| \otimes X + |10\rangle\langle 10| \otimes Z + |11\rangle\langle 11| \otimes XZ$.

Failure Example The script below is an example of failure of verification. With the option `-d`, the verifier shows processes, environments and partial traces when the verification fails. This could be useful to find equations that are necessary to verify bisimulation of configurations.

```
$ ruby verifier.rb -d scripts/epr_or_prob_err.scr

...
A's env. after trace out:
e|-["Tr", ["q1", "q2", "q3", "q6"]]
 e|-["*E0", ["qEVE"]]
  e|-["EPR", ["q3", "q4"]]
  e|-["EVE", ["qEVE"]]
```

```
nat 2;
nat m;
channel c : 2;
channel d : 1;
qvar q  : 1;
qvar q1 : 1;
qvar q2 : 1;
qvar x : 2;
qvar qE : m;
dsym EPR : 1,1;
dsym ZERO : 2;
dsym AFTER : 1,1,2;
dsym ANY : 1;
dsym EVE : m;
operator cnot : 1,1;
operator hadamard : 1;
operator measure : 1,1,2;
operator telproc : 2,1;
operator swap : 1,1;

process Tel_Proc
 ((cnot[q,q1].
   hadamard[q].
   measure[q,q1,x].
   c!x.discard(q,q1)
 ||
   c?y.telproc[y,q2].
   d!q2.discard(y)
 )/{c})
end

environment Tel_Env
 EPR[q1,q2] * ZERO[x]
 * ANY[q] * EVE[qE]
end

configuration Tel
 proc Tel_Proc
 env  Tel_Env
end

process TelSpec_Proc
 swap[q,q2].d!q2.discard(q1,x,q)
end

environment TelSpec_Env
 EPR[q1,q2] * ZERO[x]
 * ANY[q] * EVE[qE]
end

configuration TelSpec
 proc TelSpec_Proc
 env  TelSpec_Env
end

equation E1
 telproc[x,q2](measure[q,q1,x](
 hadamard[q](cnot[q,q1](
 EPR[q1,q2] * ZERO[x] * ANY[q]))))
 =
 ANY[q2] * AFTER[q,q1,x]
end

equation E2
 swap[q,q2](EPR[q1,q2] * ANY[q])
 =
 EPR[q1,q] * ANY[q2]
end
```

Figure 1: Code of quantum teleportation

```
B's env. after trace out:
e|-["Tr", ["q1", "q2", "q3", "q6"]]
 e|-["*E0", ["qEVE"]]
  e|-["EVE", ["qEVE"]]
  e|-["PROB", ["q3", "q4"]]
```

They cannot be transformed to the same expression because user-defined equation cannot be applied any more. With `-u`, the verifier only shows partial traces while `-d` shows all other information. These options will be useful to find equations that are necessary to verify bisimilarity. In this case, an equation `Tr[q3](EPR[q3,q4]])=Tr[q3](PROB[q3,q4]])` is found to be necessary.