Automated Quantum Program Verification in Dynamic Quantum Logic

Tsubasa Takagi(✉), Canh Minh Do, and Kazuhiro Ogata

Japan Advanced Institute of Science and Technology, Nomi, Japan
{tsubasa,canhdo,ogata}@jaist.ac.jp

Abstract. Dynamic Quantum Logic (DQL) has been used as a logical framework for manually proving the correctness of quantum programs. This paper presents an automated approach to quantum program verification at the cost of simplifying DQL to Basic Dynamic Quantum Logic (BDQL). We first formalize quantum states, quantum gates, and projections in bra-ket notation and use a set of laws from quantum mechanics and matrix operations to reason on quantum computation. We then formalize the semantics of BQDL and specify the behavior and desired properties of quantum programs in the scope of BDQL. Formal verification of whether a quantum program satisfies desired properties is conducted automatically through an equational simplification process. We use Maude, a rewriting logic-based specification/programming language, to implement our approach. To demonstrate the effectiveness of our automated approach, we successfully verified the correctness of five quantum protocols: Superdense Coding, Quantum Teleportation, Quantum Secret Sharing, Entanglement Swapping, and Quantum Gate Teleportation, using our support tool.

Keywords: Dynamic Quantum Logic · Quantum Programs · Quantum Protocols · Maude

1 Introduction

Quantum computing has the potential to transform various computing applications, such as cryptography [24], deep learning [8], optimization [12], and solving linear systems [17], by offering the ability to solve problems that are currently infeasible for classical computing, such as Shore's fast algorithm for integer factoring and Grover's fast algorithm for finding a datum in an unsorted database. However, quantum computing is counter-intuitive and distinct from classical computing, which makes it challenging to design and implement quantum protocols, algorithms, and programs accurately. Therefore, it is crucial to ensure their correctness through verification. While existing formal verification techniques can be used to verify that classical systems enjoy some desired properties, they cannot be directly applied to quantum systems due to the distinct principles used in quantum computing [27]. Therefore, new formal verification techniques are necessary for quantum systems.

N. Gierasimczuk and F. R. Velázquez-Quesada (Eds.): DaLí 2023, LNCS 14401, pp. 68–84, 2024.
https://doi.org/10.1007/978-3-031-51777-8_5

An extension of Quantum Logic [9] called Dynamic Quantum Logic (DQL) can be utilized to describe specifications of quantum programs. So far, various quantum protocols, such as Superdense Coding [6], Quantum Teleportation [5], Quantum Secret Sharing [19], Entanglement Swapping [28], and Quantum Gate Teleportation [14] have been verified using a DQL called the Logic of Quantum Programs (LQP) [1,2] (see [4] for a comprehensive review of DQL). However, these protocols were only verified manually by giving their correctness proofs, and it has not been known how to automate this process.

This paper presents an automated approach to quantum program verification in Basic Dynamic Quantum Logic (BDQL). BDQL is a simplified version of DQL, reflecting its essential features from an implementation perspective. We first formalize quantum states, quantum gates, and projections in bra-ket notation and use a set of laws from quantum mechanics and matrix operations to reason on quantum computation. This formalization is adopted from symbolic reasoning in [11]. The advantage of this symbolic reasoning is that we use bra-ket notation instead of explicitly complex vectors and matrices as is proposed in [22], which makes our representations more compact. Moreover, we can deal not only with concrete values but also with symbolic values for complex numbers reasoning. We then formalize the semantics of BDQL in order to describe the behavior and desired properties of quantum programs. We use Maude [10], a high-performance specification/programming language based on rewriting logic [20], to implement our approach. The symbolic reasoning on quantum computation and the semantics of BDQL are formalized by means of equations in Maude. Therefore, formal verification of quantum programs in BDQL is conducted automatically through a simplification process with respect to the equations in Maude.

Using our support tool, we successfully verify the correctness of five quantum protocols: Superdense Coding, Quantum Teleportation, Quantum Secret Sharing, Entanglement Swapping, and Quantum Gate Teleportation. This demonstrates the effectiveness of our automated approach to verify quantum programs in BDQL with symbolic reasoning adopted from [11] in practice. The support tool and case studies are available at https://github.com/canhminhdo/DQL.

2 Basic Dynamic Quantum Logic

In this section, we formulate Basic Dynamic Quantum Logic (BDQL). It is possible to describe and verify at least the five specific protocols in Sect. 3. Because the five protocols are utilized for more complex protocols, BDQL is a sufficiently expressive logic as a starting point. Further extensions of our BDQL will be required to verify other protocols in the future.

Let L_0 be a set of atomic formulas and Π_0 be a set of atomic programs. The set L of all formulas in BDQL and the set Π of all star-free regular programs are generated by simultaneous induction as follows:

$$L \ni A ::= p \mid \neg A \mid A \wedge A \mid [a]A,$$
$$\Pi \ni a ::= \mathbf{skip} \mid \mathbf{abort} \mid \pi \mid a\,;a \mid a \cup a \mid A?,$$

where $p \in L_0$ and $\pi \in \Pi_0$. The symbols **skip** and **abort** are called constant programs. The operators ;, $\cup$, and ? are called sequential composition, nondeterministic choice, and test, respectively.

The syntax of BDQL is exactly the same as that of Propositional Dynamic Logic (PDL) [16] without the Kleene star operator *. It is not strange that two different logics have the same syntax. For example, Classical Logic and Intuitionistic Logic have the same syntax but are distinguished by their semantics (or their sets of provable formulas). Similarly, the semantics of BDQL and that of the star-free fragment of PDL are different. This paper considers only star-free regular programs, leaving the addition of * to the logic in a future paper.

We define the semantics of BDQL using frames and models as usual. This kind of semantics is called Kripke (or relational) semantics.

- A quantum dynamic frame is a pair $F = (\mathcal{H}, v)$ that consists of a Hilbert space $\mathcal{H}$ and a function v from Π_0 to the set $\mathcal{U}(\mathcal{H})$ of all unitary operators on $\mathcal{H}$. The function v is called an interpretation for atomic programs.
- A quantum dynamic model is a triple $M = (\mathcal{H}, v, V)$ that consists of a quantum dynamic frame $(\mathcal{H}, v)$ and a function V from L_0 to the set $\mathcal{C}(\mathcal{H})$ of all closed subspaces of $\mathcal{H}$. The function V is called an interpretation for atomic formulas.

The definition of quantum dynamic models states that atomic formulas are interpreted as a closed subspace of a Hilbert space. This interpretation is known as the algebraic semantics for Quantum Logic [9]. The set $\mathcal{C}(\mathcal{H})$ is called a Hilbert lattice [23] because it forms a lattice with meet $X \cap Y$ and join $X \sqcup Y = (X^\perp \cap Y^\perp)^\perp$ for any $X, Y \in \mathcal{C}(\mathcal{H})$, where $^\perp$ denotes the orthogonal complement. Note that $X \sqcup Y \supseteq X \cup Y$ and $X \cup Y \notin \mathcal{C}(\mathcal{H})$ in general.

Remark 1. Usually, Kripke frames are defined as a pair (tuple) that consists of a non-empty set S and relation(s) R on S. On the other hand, the quantum dynamic frames defined above have no relation(s). However, the relations can be recovered immediately using v. That is, the family $\{R_\pi : \pi \in \Pi_0\}$ of relations on $\mathcal{H}$ is constructed by

$$R_\pi = \{(s, t) : (v(\pi))(s) = t\}$$

for each $\pi \in \Pi_0$. For this reason, we use the word "frame" for quantum dynamic frames.

The interpretation v is defined for atomic programs, and V is defined for atomic formulas. These interpretations are extended to that for star-free regular programs and formulas, respectively. For any quantum dynamic model M, the function $[\![\ ]\!]^M : L \to \mathcal{C}(\mathcal{H})$ and family $\{R_a^M : a \in \Pi\}$ of relations on $\mathcal{H}$ are defined by simultaneous induction as follows:

1. $[\![p]\!]^M = V(p)$;
2. $[\![\neg A]\!]^M$ is the orthogonal complement of $[\![A]\!]^M$;
3. $[\![A \wedge B]\!]^M = [\![A]\!]^M \cap [\![B]\!]^M$;

4. $[\![[a]A]\!]^M = \{s \in \mathcal{H} : (s,t) \in R_a^M \text{ implies } t \in [\![A]\!]^M \text{ for any } t \in \mathcal{H}\}$;
5. $R^M_{\mathbf{skip}} = \{(s,t) : s = t\}$;
6. $R^M_{\mathbf{abort}} = \emptyset$;
7. $R^M_\pi = \{(s,t) : (v(\pi))(s) = t\}$;
8. $R^M_{a;b} = \{(s,t) : (s,u) \in R_a^M \text{ and } (u,t) \in R_b^M \text{ for some } u \in \mathcal{H}\}$;
9. $R^M_{a \cup b} = R_a^M \cup R_b^M$;
10. $R^M_{A?} = \{(s,t) : P_{[\![A]\!]^M}(s) = t\}$, where $P_{[\![A]\!]^M}$ stands for the projection onto $[\![A]\!]^M$.

Theorem 1. *$[\![\,]\!]^M$ is well-defined. That is, $[\![A]\!]^M \in \mathcal{C}(\mathcal{H})$ for each $A \in L$.*

Proof. See Appendix.

The function $[\![\,]\!]^M$ and family $\{R_a^M : a \in \Pi\}$ are uniquely determined if M is given. Recall that $v(\pi)$ is a function. On the other hand, R_a^M is a relation and may not be a function due to $\cup$.

Now we can understand the meaning of each program: **skip** does nothing, **abort** forces to halt without executing subsequent programs, ; is the composition operator, $\cup$ is the non-deterministic choice operator, and ? is the quantum test operator and is used to represent a result of projective measurement (see Sect. 3.1).

Henceforth, we write $(M,s) \models A$ for the condition $s \in [\![A]\!]^M$ as usual. That is, $(M,s) \models A$ if and only if $P_{[\![A]\!]^M}(s) = s$. A formula A is said to be satisfiable (resp. valid) if $(M,s) \models A$ for some (resp. any) M and $s \in \mathcal{H}$.

Remark 2. In most modal logics, a contradiction $A \wedge \neg A$ is not satisfiable. In other words, not $(M,s) \models A \wedge \neg A$ for any s. On the other hand, $A \wedge \neg A$ is satisfiable in BDQL because $(M, \mathbf{0}) \models A \wedge \neg A$, where $\mathbf{0}$ stands for the origin (zero vector) of $\mathcal{H}$. LQP [2] chooses different semantics from that in this paper to avoid this. That is, $\mathbf{0}$ (or the corresponding subspace $\{\mathbf{0}\}$) is not a state in the semantics of LQP. Unlike LQP, we allow $\mathbf{0}$ to be a state; otherwise, our definition is ill-defined (Theorem 1 does not hold).

The following theorem gives the theoretical background for rewriting the statement of the form $(M,s) \models A$ in implementation explained in Sect. 4.

Theorem 2. *The following holds for any M and $s \in \mathcal{H}$.*

1. *$(M,s) \models A \wedge B$, if and only if $(M,s) \models A$ and $(M,s) \models B$.*
2. *$(M,s) \models [\mathbf{skip}]A$ if and only if $(M,s) \models A$.*
3. *$(M,s) \models [\mathbf{abort}]A$.*
4. *$(M,s) \models [\pi]A$ if and only if $(M,(v(\pi))(s)) \models A$.*
5. *$(M,s) \models [a\,;b]A$ if and only if $(M,s) \models [a][b]A$.*
6. *$(M,s) \models [a \cup b]A$ if and only if $(M,s) \models [a]A \wedge [b]A$.*
7. *$(M,s) \models [A?]B$ if and only if $(M, P_{[\![A]\!]^M}(s)) \models B$.*

Proof. Straightforward.

3 Application to Quantum Program Verification

This section describes the behavior and desired properties of some specific quantum programs in the language of BDQL. These properties can be verified automatically using our support tool as shown in Sect. 4 and 5.

3.1 Basic Notions

In the beginning, we briefly review quantum computation and fix our notation. We assume the readers have basic knowledge of linear algebra.

Generally speaking, quantum systems are formulated as complex Hilbert spaces. However, for quantum computation, it is enough to consider specific Hilbert spaces called qubit systems. An n-qubit system is the complex 2^n-space $\mathbb{C}^{2^n}$, where $\mathbb{C}$ stands for the complex plane. Pure states in the n-qubit system $\mathbb{C}^{2^n}$ are unit vectors in $\mathbb{C}^{2^n}$. The orthogonal basis called computational basis in the one-qubit system $\mathbb{C}^2$ is a set $\{|0\rangle, |1\rangle\}$ that consists of the column vectors $|0\rangle = (1,0)^T$ and $|1\rangle = (0,1)^T$, where ^{T} denotes the transpose operator. The linear combinations $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ and $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$ of $|0\rangle$ and $|1\rangle$ are also pure states. In general, $|\psi\rangle = c_0|0\rangle + c_1|1\rangle$ represents a pure state in the one-qubit system $\mathbb{C}^2$ provided that $|c_0|^2 + |c_1|^2 = 1$. In the two-qubit system $\mathbb{C}^4$, there are pure states that cannot be represented in the form $|\psi_1\rangle \otimes |\psi_2\rangle$ and are called entangled states, where $\otimes$ denotes the tensor product (more precisely, the Kronecker product). For example, the EPR state (Einstein-Podolsky-Rosen state) $|\mathrm{EPR}\rangle = (|00\rangle + |11\rangle)/\sqrt{2}$ is an entangled state, where $|00\rangle = |0\rangle \otimes |0\rangle$ and $|11\rangle = |1\rangle \otimes |1\rangle$.

The above notation of vectors is called bra-ket notation (also called Dirac notation). $|\psi\rangle$ is called a ket vector. A bra vector $\langle\psi|$ is defined as a row vector whose elements are complex conjugates of the elements of the corresponding ket vector $|\psi\rangle$. Observe that the matrix multiplication $|\psi\rangle\langle\psi|$ is the projection onto the subspace spanned by $|\psi\rangle$.

Quantum computation is represented by unitary operators (also called quantum gates). There are various quantum gates. For example, the Hadamard gate H and Pauli gates X, Y, and Z are typical quantum gates on the one-qubit system $\mathbb{C}^2$ and are defined as follows:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

Two typical quantum gates on the two-qubit system $\mathbb{C}^4$ are the controlled-X gate (also called the controlled NOT gate) CX and the swap gate SWAP are defined by

$$\mathrm{CX} = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X,$$
$$\mathrm{SWAP} = \mathrm{CX}(I \otimes |0\rangle\langle 0| + X \otimes |1\rangle\langle 1|)\mathrm{CX},$$

where I denotes the identity matrix of size 2×2. Measurement is a completely different process from applying quantum gates. Here, we roughly explain specific

projective measurements. For the general definition of projective measurement, see [21]. Observe that $P_0 = |0\rangle\langle 0|$ and $P_1 = |1\rangle\langle 1|$ are projections, respectively. After executing the measurement $\{P_0, P_1\}$, a current state $|\psi\rangle = c_0|0\rangle + c_1|1\rangle$ is transitioned into $P_0|\psi\rangle/|c_0| = c_0|0\rangle/|c_0|$ with probability $|c_0|^2$ and into $P_1|\psi\rangle/|c_1| = c_1|1\rangle/|c_1|$ with probability $|c_1|^2$. There is no other possibility because $|c_0|^2 + |c_1|^2 = 1$.

3.2 Standard Interpretation

To describe the quantum programs discussed in this paper, we fix

$$\begin{aligned} \Pi_0 &= \{\mathtt{H}(i), \mathtt{X}(i), \mathtt{Y}(i), \mathtt{Z}(i), \mathtt{CX}(i,j), \mathtt{SWAP}(i,j) : i,j \in \mathbb{N}, i \neq j\}, \\ L_0 &= \{p(i, |\psi\rangle), p(i, i+1, |\Psi\rangle) : i \in \mathbb{N}, |\psi\rangle \in \mathbb{C}^2, |\Psi\rangle \in \mathbb{C}^4\}, \end{aligned}$$

where $\mathbb{N}$ stands for the set of all natural numbers (including 0). Because now atomic programs and atomic formulas are restricted, we only need to consider specific interpretations called the standard interpretations $\bar{v}$ and $\overline{V}$ instead of v and V, respectively. The standard interpretations are defined below.

A function $\bar{v} : \Pi_0 \to \mathcal{U}(\mathbb{C}^{2^n})$ is called the standard interpretation on $\mathbb{C}^{2^n}$ for atomic programs if

$$\begin{aligned} \bar{v}(\mathtt{H}(i)) &= I^{\otimes i} \otimes H \otimes I^{\otimes n-i-1}, \quad \bar{v}(\mathtt{X}(i)) = I^{\otimes i} \otimes X \otimes I^{\otimes n-i-1}, \\ \bar{v}(\mathtt{Y}(i)) &= I^{\otimes i} \otimes Y \otimes I^{\otimes n-i-1}, \quad \bar{v}(\mathtt{Z}(i)) = I^{\otimes i} \otimes Z \otimes I^{\otimes n-i-1}, \end{aligned}$$

$$\begin{aligned} \bar{v}(\mathtt{CX}(i,j)) &= I^{\otimes i} \otimes |0\rangle\langle 0| \otimes I^{\otimes n-i-1} \\ &\quad + (I^{\otimes i} \otimes |1\rangle\langle 1| \otimes I^{\otimes n-i-1})(I^{\otimes j} \otimes X \otimes I^{\otimes n-j-1}), \\ \bar{v}(\mathtt{SWAP}(i,j)) &= \bar{v}(\mathtt{CX}(i,j)\ ;\ \mathtt{CX}(j,i)\ ;\ \mathtt{CX}(i,j)), \end{aligned}$$

where

$$I^{\otimes i} = \overbrace{I \otimes \cdots \otimes I}^{i}.$$

That is, under the standard interpretation, $\mathtt{H}(i)$, $\mathtt{X}(i)$, $\mathtt{Y}(i)$, $\mathtt{Z}(i)$ execute the corresponding quantum gate on the i-th qubit, $\mathtt{CX}(i,j)$ executes the Pauli gate X on the target qubit (j-th qubit) depending on the state of the control qubit (i-th qubit), and $\mathtt{SWAP}(i,j)$ swaps the i-th and j-th qubits.

A function $\overline{V} : L_0 \to \mathcal{C}(\mathbb{C}^{2^n})$ is called the standard interpretation on $\mathbb{C}^{2^n}$ for atomic formulas if

$$\begin{aligned} \overline{V}(p(i, |\psi\rangle)) &= \mathbb{C}^{2^i} \otimes \mathrm{span}\{|\psi\rangle\} \otimes \mathbb{C}^{2^{n-i-1}}, \\ \overline{V}(p(i, i+1, |\Psi\rangle)) &= \mathbb{C}^{2^i} \otimes \mathrm{span}\{|\Psi\rangle\} \otimes \mathbb{C}^{2^{n-i-2}}, \end{aligned}$$

where $\mathrm{span}\{|\psi\rangle\}$ (resp. $\mathrm{span}\{|\Psi\rangle\}$) stands for the subspace spanned by $\{|\psi\rangle\}$ (resp. $\{|\Psi\rangle\}$).

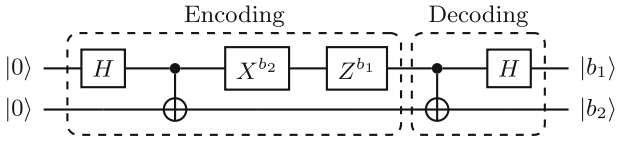

Fig. 1. Superdense Coding

In what follows, we write $\overline{M}_n$ for $(\mathbb{C}^{2^n}, \bar{v}, \overline{V})$, where the index n represents the number of qubits. In addition, we use the following abbreviation to conventionally describe quantum programs in BDQL:

$$\textbf{if } A \textbf{ then } a \textbf{ else } b \textbf{ fi} = (A? \,;\, a) \cup (\neg A? \,;\, b).$$

This program means the selection depends on the outcomes of projective measurement. That is, execute a or b depending on the result of the measurement $\{P_{[\![A]\!]^M}, P_{[\![\neg A]\!]^M}\}$. Because projective measurement occurs only in quantum computation, the behavior of this selection command is different from the usual (classical) **if then else fi** program.

3.3 Case Studies

Superdense Coding

Superdense Coding [6] allows us to transmit two classical bits using an entangled state. It consists of encoding and decoding the information. The encoding process of information 00, 01, 10, or 11 is described as follows:

$$\textbf{encode}_{00} = \mathtt{H}(0) \,;\, \mathtt{CX}(0,1), \quad \textbf{encode}_{01} = \mathtt{H}(0) \,;\, \mathtt{CX}(0,1) \,;\, \mathtt{X}(0),$$

$$\textbf{encode}_{10} = \mathtt{H}(0) \,;\, \mathtt{CX}(0,1) \,;\, \mathtt{Z}(0), \quad \textbf{encode}_{11} = \mathtt{H}(0) \,;\, \mathtt{CX}(0,1) \,;\, \mathtt{X}(0) \,;\, \mathtt{Z}(0).$$

The decoding process is described as $\textbf{decode} = \mathtt{CX}(0,1) \,;\, \mathtt{H}(0)$.

The desired property for Superdense Coding is that "the encoded information is correctly decoded." In BDQL, this property is expressed as follows:

$$(\overline{M}_2, |0\rangle \otimes |0\rangle) \models \bigwedge_{i,j \in \{0,1\}} [\textbf{encode}_{ij} \,;\, \textbf{decode}](p(0, |i\rangle) \wedge p(1, |j\rangle)).$$

Quantum Teleportation

Quantum Teleportation [5] is a protocol for teleporting an arbitrary pure state by sending two bits of classical information. The program of Quantum Teleportation is described as follows:

$$\begin{aligned} \textbf{teleport} = {} & \mathtt{H}(1) \,;\, \mathtt{CX}(1,2) \,;\, \mathtt{CX}(0,1) \,;\, \mathtt{H}(0) \\ & ; \textbf{if } p(1, |0\rangle) \textbf{ then skip else } \mathtt{X}(2) \textbf{ fi} \\ & ; \textbf{if } p(0, |0\rangle) \textbf{ then skip else } \mathtt{Z}(2) \textbf{ fi}. \end{aligned}$$

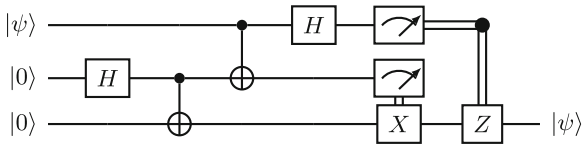

Fig. 2. Quantum Teleportation

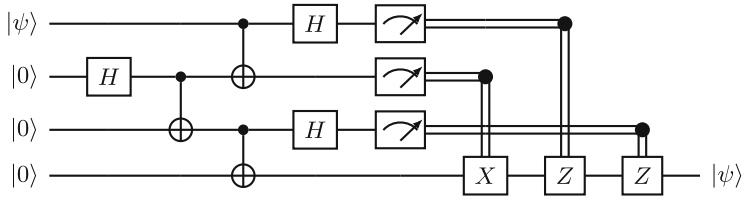

Fig. 3. Quantum Secret Sharing

The desired property of Quantum Teleportation is that "a pure state $|\psi\rangle$ is correctly teleported." In BDQL, this property is expressed as follows:

$$(\overline{M}_3, |\psi\rangle \otimes |0\rangle \otimes |0\rangle) \models [\textbf{teleport}]p(2, |\psi\rangle).$$

Quantum Secret Sharing

Quantum Secret Sharing [19] is a protocol for teleporting a pure state from a sender (Alice) to a receiver (Bob) with the help of a third party (Charlie). By this protocol, a secret pure state is shared between Alice and Bob, provided that Charlie permits it. The program of Quantum Secret Sharing is described as follows:

$$\begin{aligned}\textbf{share} = {} & \texttt{H}(1)\ ;\ \texttt{CX}(1,2)\ ;\ \texttt{CX}(0,1)\ ;\ \texttt{H}(0)\ ;\ \texttt{CX}(2,3)\ ;\ \texttt{H}(2) \\ & ;\ \textbf{if}\ p(1,|0\rangle)\ \textbf{then skip else}\ \texttt{X}(3))\ \textbf{fi} \\ & ;\ \textbf{if}\ p(0,|0\rangle)\ \textbf{then skip else}\ \texttt{Z}(3)\ \textbf{fi} \\ & ;\ \textbf{if}\ p(2,|0\rangle)\ \textbf{then skip else}\ \texttt{Z}(3)\ \textbf{fi}.\end{aligned}$$

The desired property of secret sharing is similar to that of Quantum Teleportation. In BDQL, this property is expressed as follows:

$$(\overline{M}_4, |\psi\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle) \models [\textbf{share}]p(3, |\psi\rangle)).$$

Entanglement Swapping

Entanglement Swapping [28] is a protocol for creating a new entangled state. Suppose that Alice and Bob share two entangled qubits, and Bob and Charlie

also share two different entangled qubits. After executing Entanglement Swapping, Alice's qubit and Charlie's qubit become entangled. The program of Entanglement Swapping is described as follows:

$$\begin{aligned}\textbf{entangle} = {} & \mathtt{H}(0)\ ;\ \mathtt{CX}(0,1)\ ;\ \mathtt{H}(2)\ ;\ \mathtt{CX}(2,3)\ ;\ \mathtt{CX}(1,2)\ ;\ \mathtt{H}(1) \\ & ;\ \textbf{if}\ p(2,|0\rangle)\ \textbf{then skip else}\ \mathtt{X}(3)\ \textbf{fi} \\ & ;\ \textbf{if}\ p(1,|0\rangle)\ \textbf{then skip else}\ \mathtt{Z}(3)\ \textbf{fi} \\ & ;\ \mathtt{SWAP}(1,3).\end{aligned}$$

The last $\mathtt{SWAP}(1,3)$ is executed to adjoin the remaining qubits.

The desired property of Entanglement Swapping is that "an entangled state (in this case, $|\mathrm{EPR}\rangle$) is created." In BDQL, this property is expressed as follows:

$$(\overline{M}_4, |0\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle) \models [\textbf{entangle}]p(0,1,|\mathrm{EPR}\rangle).$$

Note that $\mathtt{SWAP}(1,3)$ is needed because $p(i,i+1,|\Psi\rangle)$ is only defined for the consecutive numbers i and $i+1$. That is, the expression $p(0,3,|\mathrm{EPR}\rangle)$ is not defined.

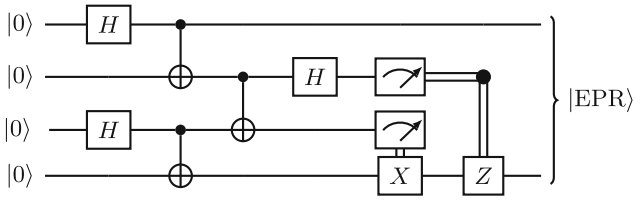

Fig. 4. Entanglement Swapping

Quantum Gate Teleportation

Quantum Gate Teleportation [14] is a protocol for teleporting a quantum gate. The program of quantum gate teleportation is described as follows:

$$\begin{aligned}\textbf{gteleport} = {} & \mathtt{H}(1)\ ;\ \mathtt{CX}(1,2)\ ;\ \mathtt{H}(3)\ ;\ \mathtt{CX}(3,4)\ ;\ \mathtt{CX}(3,2)\ ;\ \mathtt{CX}(0,1)\ ;\ \mathtt{H}(0)\ ;\ \mathtt{CX}(4,5)\ ;\ \mathtt{H}(4) \\ & ;\ \textbf{if}\ p(0,|0\rangle)\ \textbf{then skip else}\ \mathtt{Z}(2)\ ;\ \mathtt{Z}(3)\ \textbf{fi} \\ & ;\ \textbf{if}\ p(1,|0\rangle)\ \textbf{then skip else}\ \mathtt{X}(2)\ \textbf{fi} \\ & ;\ \textbf{if}\ p(5,|0\rangle)\ \textbf{then skip else}\ \mathtt{X}(2)\ ;\ \mathtt{X}(3)\ \textbf{fi} \\ & ;\ \textbf{if}\ p(4,|0\rangle)\ \textbf{then skip else}\ \mathtt{Z}(3)\ \textbf{fi}.\end{aligned}$$

The desired property of Quantum Gate Teleportation is that "a quantum gate (in this case, CX) is correctly teleported." In BDQL, this property is expressed as follows:

$$(\overline{M}_6, |\psi\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle \otimes |\psi'\rangle) \models [\textbf{gteleport}]p(3,4,\mathrm{CX}(|\psi'\rangle \otimes |\psi\rangle)).$$

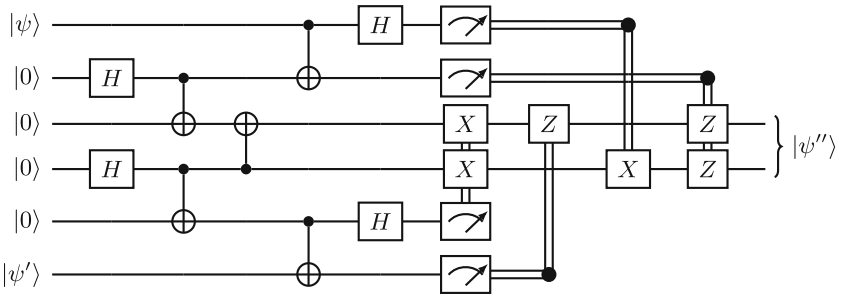

Fig. 5. Quantum Gate Teleportation ($|\psi''\rangle = \mathrm{CX}(|\psi'\rangle \otimes |\psi\rangle)$))

4 Implementation of Basic Dynamic Quantum Logic

This section describes the implementation of BDQL in Maude [10], a specification/programming language based on rewriting logic [20]. Hence, the notations used in this section follow the Maude syntax.

4.1 Syntax of Basic Dynamic Quantum Logic

We formalize pure states in a program as a collection of qubits associated with indices that start from 0 to $N-1$, where N is the total number of qubits. Hence, we can flexibly refer to a specific part of a quantum state using indices. We adopt this formalization to reason on quantum computation from [11].

We define two sorts `AtomicProg` and `Prog` for atomic programs Π_0 and star-free regular programs Π in BDQL, respectively, where `AtomicProg` is a subsort of `Prog`. We also define several operators for atomic programs as follows:

```
sorts AtomicProg Prog .
subsort AtomicProg < Prog .
ops I(_) H(_) X(_) Y(_) Z(_) : Nat -> AtomicProg [ctor] .
op CX(_,_) : Nat Nat -> AtomicProg [ctor] .
```

where `I(_)`, `H(_)`, `X(_)`, `Y(_)`, `Z(_)` operators take a natural number as input, denoting the index of a qubit of a pure state on which the quantum gates I, H, X, Y, Z, will be applied, respectively; and `CX(_,_)` operator takes two natural numbers as inputs, denoting the indices of two qubits of a pure state on which CX will be applied. These operators serve as the constructors of atomic programs with the `ctor` attribute.

We use several operators to define star-free regular programs in BDQL as follows:

```
ops abort skip : -> Prog [ctor] .
op _;_ : Prog Prog -> Prog [ctor assoc id: skip prec 25] .
op _U_ : Prog Prog -> Prog [ctor] .
op _? : Formula -> Prog [ctor prec 24] .
```

where all operators follow the definition of star-free regular programs in BDQL shown in Sect. 2; besides that, the `skip` operator also denotes an empty program; `ctor`, `assoc`, `id:_`, and `prec_` are operator attributes for a constructor, associativity, an identity element, and operator precedence, respectively.

We define two sorts `AtomicFormula` and `Formula` for atomic formulas L_0 and general formulas L in BDQL, respectively, where `AtomicFormula` is a subsort of `Formula`. We also define several operators for constructing formulas in DQL as follows:

```
sorts AtomicFormula Formula .
subsort AtomicFormula < Formula .
op P(_,_) : Nat Matrix -> AtomicFormula [ctor] .
op P(_,_,_) : Nat Nat Matrix -> AtomicFormula [ctor] .
op neg_ : Formula -> Formula .
op _/\_ : Formula Formula -> Formula [ctor comm assoc] .
op [_]_ : Prog Formula -> Formula [ctor] .
```

where `Matrix` is a family sort of quantum states, quantum gates, and projections because they can be expressed in terms of matrices. The `P(_,_)` and `P(_,_,_)` operators are atomic formulas representing projections of the forms $p(i, |\psi\rangle)$ and $p(i, j, |\Psi\rangle)$, respectively (see Sect. 3.2). The other operators follow the definition of formulas in BDQL as shown in Sect. 2.

The `if-then-else-fi` command corresponding to the program **if then else fi** is implemented as follows:

```
op if_then_else_fi : Formula Prog Prog -> Prog .
eq if F1:Formula then P1:Prog else P2:Prog fi
= (F1:Formula ? ; P1:Prog) U ((neg F2:Formula) ? ; P2:Prog) .
```

4.2 Semantics of Basic Dynamic Quantum Logic

The semantics of $(M, s) \models A$ in BDQL is represented by the term `s |= A` of sort `Judgment` with the following operator.

```
sort Judgment .
op _|=_ : QState Formula -> Judgment .
```

where sort `QState` represents quantum states (more precisely, pure states).

The satisfiability of $A \wedge B$ in BDQL is determined by that of A and B (Theorem 2), each of which is represented by a judgment. Hence, we need a sort to represent a set of judgments as follows:

```
sort JudgmentSet .  subsort Judgment < JudgmentSet .
op emptyJS : -> JudgmentSet [ctor] .
op _/\_ : JudgmentSet JudgmentSet -> JudgmentSet [ctor assoc
    comm id: emptyJS] .
```

where `emptyJS` is the empty set of judgments, and the operator `_/_` serves as the constructor of the set (`ctor`), is associative (`assoc`), commutative (`comm`), and has the empty set as an identity element (`id: emptyJS`).

Now, we implement the semantics of BDQL using equations that simplify a judgment `s |= A` into the set of judgments as follows:

```
vars PROG PROG' : Prog .          vars Q Q' : QState .
vars N N1 N2 : Nat .              var M : Matrix .
vars Phi Psi : Formula .
ceq Q |= P(N, M) = emptyJS if (Q).P(N, M) == Q .
ceq Q |= P(N1, N2, M) = emptyJS if (Q).P(N1, N2, M) == Q .
eq neg P(N:Nat, |0>) = P(N:Nat, |1>) .
eq neg P(N:Nat, |1>) = P(N:Nat, |0>) .
eq Q |= Phi /\ Psi = (Q |= Phi) /\ (Q |= Psi) .
eq Q |= [skip] Phi = Q |= Phi .
eq Q |= [abort] Phi = emptyJS .
eq Q |= [I(N)] Phi = Q |= Phi .
ceq Q |= [H(N)] Phi = Q' |= Phi if Q' := (Q).H(N) .
ceq Q |= [X(N)] Phi = Q' |= Phi if Q' := (Q).X(N) .
ceq Q |= [Y(N)] Phi = Q' |= Phi if Q' := (Q).Y(N) .
ceq Q |= [Z(N)] Phi = Q' |= Phi if Q' := (Q).Z(N) .
ceq Q |= [CX(N1,N2)] Phi = Q' |= Phi if Q' := (Q).CX(N1,N2) .
ceq Q |= [PROG' ; PROG] Phi = Q |= [PROG']([PROG] Phi)
if PROG' =/= nil /\ PROG =/= nil .
eq Q |= [PROG' U PROG] Phi
= (Q |= [PROG'] Phi) /\ (Q |= [PROG] Phi) .
ceq Q |= [P(N,M)?] Phi = Q' |= Phi if Q' := (Q).P(N,M) .
```

where `var` and `vars` keywords are used to declare variables of some sorts. The first two equations define the semantics of the atomic formulas $p(i, |\psi\rangle)$ and $p(i, i+1, |\Psi\rangle)$, where i denotes the index at which the projections will take place and $|\psi\rangle, |\Psi\rangle$ are used to construct their projection operators in the forms of $|\psi\rangle\langle\psi|, |\Psi\rangle\langle\Psi|$, respectively. Recall that $(M, |\psi\rangle) \models p(i, |\psi\rangle)$ if and only if $P_{[\![p(i,|\psi\rangle)]\!]^M}(|\psi\rangle) = |\psi\rangle$. The next two equations define the negation of atomic formulas $p(i, |0\rangle)$ and $p(i, |1\rangle)$. It is not necessary to implement the negation of the other formulas for conducting the experiments in Sect. 5. The next equation reflects the semantics of conjunction. Based on Theorem 2, the remaining equations simulate **skip**, **abort**, quantum gates (I, H, X, Y, Z, and CX), ; (composition), $\cup$ (choice), and ? (test). Note that $A?$ is implemented only for $A = p(i, |\psi\rangle)$ because the other more complex test operators are not needed for conducting the experiments in Sect. 5. For the sake of simplicity, we do not mention how we implement the behavior of quantum gates and projections in detail to make the paper concise.

Let us suppose that $E_{\mathtt{SR}}$ and $E_{\mathtt{BDQL}}$ are the sets of equations used for symbolic reasoning on quantum computation adopted from [11] and the semantics of BDQL specified in Maude, respectively. Now we have enough facilities to check whether $(M, s) \models A$ by simplifying `s |= A` $\rightarrow^*_{E_{\mathtt{SR}} \cup E_{\mathtt{BDQL}}}$ `emptyJS`. Indeed, $(M, s) \models A$ if `s |= A` is simplified to `emptyJS` with respect to $E_{\mathtt{SR}} \cup E_{\mathtt{BDQL}}$.

Table 1. Experimental results with our support tool for the five case studies

Protocol	Qubits	Rewrite Steps	Time
Superdense Coding	2	2,659	1 ms
Quantum Teleportation	3	2,558	1 ms
Quantum Secret Sharing	4	7,139	3 ms
Entanglement Swapping	4	5,344	2 ms
Quantum Gate Teleportation	6	56,901	37 ms

5 Experiments

This section shows how to use our support tool to verify Quantum Teleportation in Maude as an example and similarly for other protocols, which can be fully found at https://github.com/canhminhdo/DQL. Subsequently, we provide the experimental results for five protocols used in the experiments.

Let `TELEPORT` be the specification of Quantum Teleportation, `initQState` be the initial state for `TELEPORT` and `qubitAt` be the function to get a single qubit at some index. We can verify the correctness of Quantum Teleportation with our support tool using the **`reduce`** command in Maude as follows:

```
reduce in TELEPORT : initQState |= [
   H(1) ; CX(1, 2) ; CX(0, 1) ; H(0) ;
   if P(1, |0>) then skip else X(2) fi ;
   if P(0, |0>) then skip else Z(2) fi
] P(2, qubitAt(initQState, 0)) .
```

The `reduce` command will conduct the simplification process with respect to the equations specified in our tool automatically. The command returns `emptyJS` in just a few moments, and thus the correctness of Quantum Teleportation is verified using our support tool, the implementation of BDQL in Maude, where the first qubit at index 0 of the initial quantum state is teleported correctly in the third qubit at index 2 of the final quantum state.

We conducted experiments on an iMac that carries a 4 GHz microprocessor with eight cores and 32 GB memory of RAM. The experimental results are shown in Table 1. We successfully verified the correctness of Superdense Coding, Quantum Teleportation, Quantum Secret Sharing, Entanglement Swapping, and Quantum Gate Teleportation according to the properties described in Sect. 3. For all case studies from two to six qubits used, we can quickly verify their correctness in just a few moments using our support tool, although the number of rewrite steps involved is quite large. Without the aid of computer programs, such as our support tool, these results would have been almost impossible. This demonstrates the effectiveness of our automated approach for verifying quantum programs in BDQL using the symbolic approach adopted from [11].

6 Related Work

Quantum Hoare Logic (QHL) by [25] was designed and intended to be a quantum counterpart of Hoare Logic. From the perspective of logic, BDQL can express more fundamental components of quantum programs compared to QHL: the **if** $\cdots$ **fi** statement that represents a non-deterministic measurement cannot be divided anymore in QHL. On the other hand, BDQL can express its non-deterministic feature explicitly using the choice operator $\cup$. Also, QHL lacks the test operator in its syntax.

In this paper, we chose Maude as our implementation language. On the other hand, [13] was implemented in PRISM (Probabilistic symbolic model checker) for verifying quantum protocols. Unlike our approach, [13] needed to enumerate states and calculate the state transitions in advance and then encode them into a specification. In contrast, our approach did not require such enumeration of states in advance because we formalized the quantum computation and the semantics of BDQL by means of equations, and the verification problem is conducted automatically through an equational simplification process in Maude.

7 Conclusions and Future Work

We have presented the implementation of BDQL, a simplified version of DQL, in Maude for quantum program verification. The symbolic reasoning from [11] is adopted, and we have formalized the semantics of BDQL by means of equations. The verification problem is simplified using the `reduce` command in Maude with respect to the equations specified in our support tool. Using our support tool, we have successfully verified the five quantum programs. This demonstrates the effectiveness of our automated approach for verifying quantum programs in BDQL using symbolic reasoning.

At least two future extensions remain to be addressed. That is, our tool (and BDQL) is limited in that (I) it cannot deal with programs including the Kleene star operator (iteration operator) and that (II) it cannot deal with quantitative properties regarding measurement probability. As to (I), it is significant to extend our tool so that it can deal with the Kleene star operator for expressing quantum loop programs [26]. As to (II), a DQL called the Probabilistic Logic of Quantum Programs (PLQP) that can express the quantitative properties has been proposed and applied to formal verification of the quantum search algorithm, Quantum Leader Election, and the BB84 quantum key distribution protocol [3,7]. However, their verification was done manually, and their automation by a tool is still an open problem.

Acknowledgements. The authors are grateful to the anonymous reviewers for their valuable feedback. The research was supported by JAIST Research Grant for Fundamental Research. The research of the first author was supported by Grant-in-Aid for JSPS Research Fellow Grant Number JP22KJ1483. The research of the second and the third authors was supported by JST SICORP Grant Number JPMJSC20C2 and JSPS KAKENHI Grant Number JP24H03370.

Appendix

Proof of Theorem 1

Before embarking on the proof of Theorem 1, we show a lemma.

Lemma 1. *The following holds for any M:*

1. $[\![[\mathbf{skip}]A]\!]^M = [\![A]\!]^M$;
2. $[\![[\mathbf{abort}]A]\!]^M = \mathcal{H}$;
3. $[\![[\mathbf{abort}; b]A]\!]^M = [\![[\mathbf{abort}]A]\!]^M$;
4. $[\![[\mathbf{skip}; b]A]\!]^M = [\![[b]A]\!]^M$;
5. $[\![[(a\ ;\ b)\ ;\ c]A]\!]^M = [\![[a\ ;\ (b\ ;\ c)]A]\!]^M$;
6. $[\![[(a \cup b)\ ;\ c]A]\!]^M = [\![(a\ ;\ c) \cup (b\ ;\ c)]A]\!]^M$;
7. $[\![[a\ ;\ b]A]\!]^M = [\![[a][b]A]\!]^M$;
8. $[\![[a \cup b]A]\!]^M = [\![[a]A]\!]^M \cap [\![[b]A]\!]^M$;
9. $[\![[B?]A]\!]^M = [\![B \to A]\!]^M \in \mathcal{C}(\mathcal{H})$, *where $B \to A$ denotes the Sasaki hook [18] defined as $\neg(A \wedge \neg(A \wedge B))$.*

Proof. 1 to 8 are easy to show. For 9, some knowledge of Hilbert space theory is required. Observe that $[\![[B?]A]\!]^M$ is the inverse image $P^{-1}_{[\![B]\!]^M}([\![A]\!]^M)$ of $[\![A]\!]^M$ under $P_{[\![B]\!]^M}$. That is,

$$\begin{aligned}[\![[B?]A]\!]^M &= P^{-1}_{[\![B]\!]^M}([\![A]\!]^M) = \{s \in \mathcal{H} : P_{[\![B]\!]^M}(s) \in [\![A]\!]^M\} \\ &= \{s \in \mathcal{H} : P_{[\![A]\!]^M} P_{[\![B]\!]^M}(s) = P_{[\![B]\!]^M}(s)\}.\end{aligned}$$

Therefore, $[\![[B?]A]\!]^M = [\![B \to A]\!]^M \in \mathcal{C}(\mathcal{H})$ (see [15]).

We use Lemma 1 to prove Theorem 1 without mentioning it.

Proof. We prove by simultaneous structural induction on formulas in BDQL and star-free regular programs. The case $A = p \in L_0$ is immediate. The cases $A = \neg B$ and $A = B \wedge C$ follow from the basic fact in Hilbert space theory. Thus, we only discuss the case $A = [a]B$.

Case 1 $a = \mathbf{skip}$. We have $[\![[a]B]\!]^M = [\![B]\!]^M \in \mathcal{C}(\mathcal{H})$ by the induction hypothesis $[\![B]\!]^M \in \mathcal{C}(\mathcal{H})$.

Case 2 $a = \mathbf{abort}$. We have $[\![[a]B]\!]^M = \mathcal{H} \in \mathcal{C}(\mathcal{H})$.

Case 3 $a = \pi \in \Pi_0$. Observe that $[\![[a]B]\!]^M$ is the inverse image of $[\![B]\!]^M$ under $v(a)$. In other words, $[\![[a]B]\!]^M$ is the image $(v(a)^\dagger)([\![B]\!]^M)$ of $[\![B]\!]^M$ under the adjoint operator $v(a)^\dagger$ of $v(a)$. Let $X^\perp$ be the orthogonal complement of a subspace X of $\mathcal{H}$, and write $X^{\perp\perp}$ for $(X^\perp)^\perp$. Recall that $X \in \mathcal{C}(\mathcal{H})$ if and only if $X^{\perp\perp} = X$. By the induction hypothesis $[\![B]\!]^M \in \mathcal{C}(\mathcal{H})$,

$$\begin{aligned}([\![[a]B]\!]^M)^{\perp\perp} &= ((v(a)^\dagger)([\![B]\!]^M))^{\perp\perp} = ((v(a)^\dagger)(([\![B]\!]^M)^\perp))^\perp \\ &= (v(a)^\dagger)(([\![B]\!]^M)^{\perp\perp}) = (v(a)^\dagger)([\![B]\!]^M) = [\![[a]B]\!]^M.\end{aligned}$$

Consequently, $[\![[a]B]\!]^M \in \mathcal{C}(\mathcal{H})$.

Case 4 $a = b\,;c$. We further split the case with respect to b.

Case 4.1 $b = \mathbf{skip}$. $[\![[a]B]\!]^M = [\![[c]B]\!]^M \in \mathcal{C}(\mathcal{H})$ by the induction hypothesis $[\![[c]B]\!]^M \in \mathcal{C}(\mathcal{H})$.

Case 4.2 $b = \mathbf{abort}$. $[\![[a]B]\!]^M = [\![[\mathbf{abort}]B]\!]^M = \mathcal{H} \in \mathcal{C}(\mathcal{H})$.

Case 4.3 $b = \pi$. $[\![[a]B]\!]^M = [\![[\pi][c]B]\!]^M$. By the induction hypothesis, $[\![[c]B]\!]^M \in \mathcal{C}(\mathcal{H})$. Thus, it follows from the similar argument of case 3 above that $[\![[a]B]\!]^M \in \mathcal{C}(\mathcal{H})$.

Case 4.4 $b = b_1\,;b_2$.

$$[\![[a]B]\!]^M = [\![[b_1\,;(b_2\,;c)]B]\!]^M = [\![[b_1][b_2\,;c]B]\!]^M \in \mathcal{C}(\mathcal{H})$$

by the induction hypothesis $[\![[b_1][b_2\,;c]B]\!]^M \in \mathcal{C}(\mathcal{H})$.

Case 4.5 $b = b_1 \cup b_2$.

$$[\![[a]B]\!]^M = [\![[(b_1\,;c) \cup (b_2\,;c)]B]\!]^M = [\![[b_1\,;c]B]\!]^M \cap [\![[b_2\,;c]B]\!]^M \in \mathcal{C}(\mathcal{H})$$

by the induction hypothesis $[\![[b_1\,;c]B]\!]^M, [\![[b_2\,;c]B]\!]^M \in \mathcal{C}(\mathcal{H})$.

Case 4.6 $b = C?$.

$$[\![[a]B]\!]^M = [\![[C?][c]B]\!]^M = [\![C \to [c]B]\!]^M \in \mathcal{C}(\mathcal{H})$$

by the induction hypothesis $[\![C]\!]^M, [\![[c]B]\!]^M \in \mathcal{C}(\mathcal{H})$.

Case 5 $a = b \cup c$. We have $[\![[a]B]\!]^M = [\![[b]B]\!]^M \cap [\![[c]B]\!]^M \in \mathcal{C}(\mathcal{H})$ by the induction hypothesis $[\![[b]B]\!]^M, [\![[c]B]\!]^M \in \mathcal{C}(\mathcal{H})$.

Case 6 $a = C?$. We have $[\![[C?]B]\!]^M = [\![C \to B]\!]^M \in \mathcal{C}(\mathcal{H})$ by the induction hypothesis $[\![B]\!]^M, [\![C]\!]^M \in \mathcal{C}(\mathcal{H})$.

References

1. Akatov, D.: The logic of quantum program verification. Master's thesis, Oxford University (2005)
2. Baltag, A., Smets, S.: LQP: the dynamic logic of quantum information. Math. Struct. Comput. Sci. **16**(3), 491–525 (2006)
3. Baltag, A., Bergfeld, J., Kishida, K., Sack, J., Smets, S., Zhong, S.: PLQP & company: decidable logics for quantum algorithms. Int. J. Theor. Phys. **53**(10), 3628–3647 (2014)
4. Baltag, A., Smets, S.: Reasoning about quantum information: an overview of quantum dynamic logic. Appli. Sci. **12**(9) (2022)
5. Bennett, C.H., Brassard, G., Crépeau, C., Jozsa, R., Peres, A., Wootters, W.K.: Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. Phys. Rev. Lett. **70**, 1895–1899 (1993)
6. Bennett, C.H., Wiesner, S.J.: Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. Phys. Rev. Lett. **69**, 2881–2884 (1992)
7. Bergfeld, J.M., Sack, J.: Deriving the correctness of quantum protocols in the probabilistic logic for quantum programs. Soft. Comput. **21**(6), 1421–1441 (2017)
8. Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., Lloyd, S.: Quantum machine learning. Nature **549**(7671), 195–202 (2017)
9. Birkhoff, G., von Neumann, J.: The logic of quantum mechanics. Ann. Math. **57**(4), 823–843 (1936)

10. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L. (eds.): All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic. LNCS, vol. 4350. Springer (2007). https://doi.org/10.1007/978-3-540-71999-1
11. Do, C.M., Ogata, K.: Symbolic model checking quantum circuits in maude. In: The 35th International Conference on Software Engineering and Knowledge Engineering, SEKE 2023 (2023)
12. Farhi, E., Goldstone, J., Gutmann, S., Lapan, J., Lundgren, A., Preda, D.: A quantum adiabatic evolution algorithm applied to random instances of an np-complete problem. Science **292**(5516), 472–475 (2001)
13. Gay, S., Nagarajan, R., Papanikolaou, N.: Probabilistic model-checking of quantum protocols. arXiv preprint quant-ph/0504007 (2005)
14. Gottesman, D., Chuang, I.L.: Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations. Nature **402**(6760), 390–393 (1999)
15. Hardegree, G.M.: The conditional in quantum logic. Synthese, 63–80 (1974)
16. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press (2000)
17. Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for linear systems of equations. Phys. Rev. Lett. **103**, 150502 (2009)
18. Herman, L., Marsden, E.L., Piziak, R.: Implication connectives in orthomodular lattices. Notre Dame J. Formal Logic **16**(3), 305–328 (1975)
19. Hillery, M., Bužek, V., Berthiaume, A.: Quantum secret sharing. Phys. Rev. A **59**, 1829–1834 (1999)
20. Meseguer, J.: Twenty years of rewriting logic. J. Log. Algebraic Methods Program **81**(7–8), 721–781 (2012)
21. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press (2011)
22. Paykin, J., Rand, R., Zdancewic, S.: Qwire: a core language for quantum circuits. SIGPLAN Not. **52**(1), 846–858 (2017)
23. Rédei, M.: Quantum logic in algebraic approach. FTPH, vol. 91. Springer (1998). https://doi.org/10.1007/978-94-015-9026-6
24. Shor, P.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124–134 (1994)
25. Ying, M.: Floyd-hoare logic for quantum programs. ACM Trans. Program. Lang. Syst. (TOPLAS) **33**(6), 1–49 (2012)
26. Ying, M., Feng, Y.: Quantum loop programs. Acta Informatica **47**(4), 221–250 (2010)
27. Ying, M., Feng, Y.: Model checking quantum systems – a survey (2018)
28. Żukowski, M., Zeilinger, A., Horne, M.A., Ekert, A.K.: "Event-ready-detectors" Bell experiment via entanglement swapping. Phys. Rev. Lett. **71**, 4287–4290 (1993)