Proceedings of the 2nd International Workshop on Formal Analysis and Verification of Post-Quantum Cryptographic Protocols (FAVPQC), 2023

Santiago Escobar[1], Ayoub Otmani[2], Sedat Akleylek[3,4] and Kazuhiro Ogata[5]

ISBN: 978-4-903092-62-1

Preface

Post-quantum cryptographic protocols refer to those replacements of classical cryptographic protocols as a precaution against future attacks from quantum computers. This has been motivated by the fact that the public-key cryptosystems used today will be no longer secure under large-scale quantum computers, which are promisingly becoming available in the near future because of the huge research and development investment. Significant efforts have been spent to standardize post-quantum cryptographic primitives and protocols, especially after the Post-quantum Cryptography Standardization Project started by NIST (https://csrc.nist.gov/projects/post-quantum-cryptography). Therefore, security analysis/verification of those post-quantum cryptographic protocols is an important factor in the construction of final secure cryptosystems. FAVPQC is organized as an event for researchers all over the world to report and discuss research related to the above-mentioned issue.

Following the previously successful FAVPQC 2022, in 2023, the second workshop has been held in Brisbane, Australia. The workshop is a satellite event of the 24th International Conference on Formal Engineering Methods (ICFEM 2023). There were six papers submitted to the workshop and at least three reviewers who were PC members were assigned to each paper. Through the standard peer-review process, all six papers were accepted for presentation at the workshop. Five papers were presented physically at the venue, while one paper was presented online. The six papers are included in this volume. Note that any PC members (including PC co-chairs) who are co-authors of a paper was never involved in the peer-review process of the paper.

Program Committee

Sedat Akleylek, Ondokuz Mayis University, Turkey & University of Tartu, Estonia (co-chair)
Christophe Chareton, LORIA-CELLO, France
Canh Minh Do, Japan Advanced Institute of Science and Technology, Japan
Santiago Escobar, Universitat Politecnica de Valencia, Spain (co-chair)
Daniel Gaina, Kyushu University, Japan
Catherine Meadows, Naval Research Laboratory, USA
Paolo Modesti, Teesside University, UK
Masaki Nakamura, Toyama Prefectural University, Japan
Kazuhiro Ogata, Japan Advanced Institute of Science and Technology, Japan (co-chair)
Ayoub Otmani, University of Rouen Normandie, France (co-chair)
Adrian Riesco, Universidad Complutense de Madrid, Spain
Min Zhang, East China Normal University, China

Organization Committee

Sedat Akleylek, Ondokuz Mayis University, Turkey & University of Tartu, Estonia
Santiago Escobar, Polytechnic University of Valencia, Spain
Kazuhiro Ogata, Japan Advanced Institute of Science and Technology, Japan
Ayoub Otmani, University of Rouen Normandie, France

Publicity Committee

Duong Dinh Tran, Japan Advanced Institute of Science and Technology, Japan (chair)

Editors' affiliations and emails

[1]*Polytechnic University of Valencia, Spain*
[2]*University of Rouen Normandie, France*
[3]*Ondokuz Mayis University, Turkey*
[4]*University of Tartu, Estonia*
[5]*Japan Advanced Institute of Science and Technology, Japan*

sescobar@upv.es (S. Escobar);
ayoub.otmani@univ-rouen.fr (A. Otmani);
sedat.akleylek@bil.omu.edu.tr (S. Akleylek);
ogata@jaist.ac.jp (K. Ogata)

Table of Contents

Formal specification of the post-quantum signature scheme FALCON in Maude

Víctor García[1,*], Santiago Escobar[1] and Kazuhiro Ogata[2]

[1]Universitat Politècnica de València (UPV), Camí de Vera, s/n, 46022 València, Valencia, Spain
[2]Japan Advanced Institute of Science and Technology (JAIST), Ishikawa 923–1292, Japan

Abstract

Cryptography is an important piece in any communication system. Digital signatures are a part of cryptography that ensures the authenticity and integrity of digital assets, vital properties for any secure communication. Due to fast advances in post-quantum technologies, the National Institute of Standards and Technologies started the Post-Quantum Cryptography project to standardise new algorithms and protocols that are secure against quantum attackers. One of the finalists is the signature scheme FALCON. We present the first formal specification, in the high-performance language Maude, of FALCON. For this purpose, we use an existing framework, originally aimed to formally specify and analyse post quantum key encapsulation mechanism. With the infrastructure provided by the framework, we encode a symbolic model of the signature scheme's behaviour and simulate an execution trace.

Keywords

Formal specification, Post-Quantum, Signature scheme, FALCON, Maude

1. Introduction

Security is a basic requirement in any information system today, where cryptography has an important role in fulfilling such needs. A building block of cryptography is a digital signature, a cryptography element that provides authentication and integrity protection to the system where it is enforced. The security of the most used digital signature schemes, e.g. Rivest-Shamir-Adleman (RSA) [1] and Elliptic Curve Digital Signature Algorithm (ECDSA) [2], is based on the hardness of solving computational problems hard to solve for classic computers, such as the prime factorization and discrete logarithm problems. Some algorithms could provide solutions to these hard computational problems. One example is Shor's prime factorization algorithm [3] when operating on computers with quantum computing capabilities.

With the race for quantum supremacy at full throttle there is a need for new secure schemes and protocols that are quantum-resistant. The National Institute of Standards and Technology (NIST) launched the Post-Quantum Cryptography (PQC) project to encourage the development

FAVPQC 2023: 2nd International Workshop on Formal Analysis and Verification of Post-Quantum Cryptographic Protocols, November 21, 2023, Brisbane, Australia

*Corresponding author.
vicgarval@upv.es (V. García); sescobar@upv.es (S. Escobar); ogata@jaist.ac.jp (K. Ogata)
https://v1ct0r-byte.github.io/ (V. García); http://personales.upv.es/sanesro/ (S. Escobar); http://www.jaist.ac.jp/~ogata/ (K. Ogata)
0000-0003-0681-1130 (V. García); 0000-0002-3550-4781 (S. Escobar); 0000-0002-4441-3259 (K. Ogata)

of new protocols resilient to adversaries with access to quantum computing power. So far, at the time of writing this paper, there have been 4 rounds, where different Public-key Encryption and Key-establishment Algorithms (KEMs) and Digital Signature Algorithms (DSAs) were submitted for standardization. The first selection of finalists took place in round 3, where CRYSTALS-Kyber was the only selected candidate to represent the KEMs section. Moreover, CRYSTALS-Dilithium, FALCON and SPHINCS+ were the finalists for DSAs. From the three finalists in the DSAs section we focus on FALCON [4].

In this paper, we present the first steps towards modelling signature schemes by reusing and adapting a framework previously developed for Key Encapsulation Mechanisms in [5]. Specifically, we show how the framework is adapted to signature schemes and how FALCON is described in this new framework, obtaining an executable symbolic model that simulates execution traces. All the specifications presented in this paper can be found in the GitHub repository at: https://github.com/v1ct0r-byte/PQC-in-Maude/tree/PQ-SIG.

Outline: The rest of the paper is structured as follows. Section 2 lists a series of formal analysis tools/techniques over post-quantum protocols. Section 3 explains basic concepts on Maude, the framework we work on and the basic idea of signature schemes. Section 4 presents the signature scheme detailing the steps in each algorithm. Section 5 settles assumptions for our symbolic model and displays the Maude specification in greater detail, focusing on the specification of FALCON. Section 6 gives some concluding remarks and future work.

2. Related Work

Advances in protocol security analysis have been made in the field of cryptography. One interesting idea is the one proposed at [6], where the author explains several examples of formal specification of protocols and introduces and explains the symbolic and computational model analysis approaches. In [7], the authors explore the current literature and papers on both symbolic and computational analysis of protocols. In this survey, they analyze the results by combining both types of analysis. This proposal was initially made by [8] to close the gap between both lines of protocol verification.

Among the various symbolic protocol analysis tools available, we have Maude-NPA [9, 10], related to the programming language Maude [11, 12, 13]. Maude-NPA has a theoretical basis on rewriting logic, unification and narrowing and performs a backwards search from a final attack state to determine whether or not it is reachable from an initial state. Some symbolic tools, such as ProVerif [14, 15], are based on an abstract representation of a protocol using Horn clauses. The verification of security properties is done by reasoning on these representative clauses. Other symbolic tools, such as Tamarin [16, 17], are based on constraint solving to perform an exhaustive, symbolic search for execution traces. Furthermore, other symbolic tools such as Scyther [18] or CPSA [19] attempt to enumerate all the essential parts of the different possible executions of a protocol. Also, AKISS [20] or the DEEPSEC prover [21] are other tools mostly used to decide equivalence properties.

Some related work can be found for the symbolic security analysis of protocols with quantum features. For example, the authors of [22] build a model of IKEv2 on a classical setting and

then perform some analysis on it for seven properties, all of it using the Tamarin prover. With this first symbolic analysis, they prove their model to be correct and corroborate previous results by other authors. Later, they extended the model with the improvements included in the latest extension of IKEv2 in order to include a quantum-resistant key exchange. With this extension, they perform a new analysis, where all properties hold, verifying the security of the new extension.

Another paper performing symbolic analysis of post-quantum protocols is [23]. The authors use the recently published tool SAPIC$^+$ to analyze the Ephemeral Diffie Hellman Over COSE (EDHOC) protocol, on its 12th draft version. With SAPIC$^+$ they can automatically transform their model to a suitable one in Tamarin, ProVerif or DEEPSEC respective syntax. This allows the authors to take advantage of the strength of each tool to perform analysis over their modular composed model of the protocol. With the analysis, they discover several flaws and report them to the team of EDHOC. The authors proposed some fixes, validated them and the proposal was accepted into the 14th draft version of the protocol.

In [24] a variant for a handshake protocol from the WireGuard VPN protocol with post-quantum capabilities is presented. They perform such adaptation by replacing the previous Diffie-Hellman-based handshake with key-encapsulation mechanisms. The authors verify the security of their proposal with symbolic and computational proofs. On the one hand, the symbolic proofs verify more security properties than the computational proofs and are computer-verified. On the other hand, computational proofs give stronger security guarantees as the proof makes less idealizing assumptions.

The closest works to our contributions in this paper are [25, 26, 27, 28, 5]. [25] provides a first approximation on the symbolic specification of post-quantum protocols in Maude-NPA. The authors decided to specify the Post-Quantum TLS protocol primitives and execution trace. This type of work is interesting and necessary to us because it demonstrates the capability of Maude-NPA, and Maude, to verify more advanced schemes automatically. In the extended version, [26], the authors report some of the experimental results of formal verification/analysis over Transport Layer Security using a parallel version of Maude-NPA. [27, 28] present the first symbolic security analyses of a collection of post-quantum protocols. These analyses served as guidelines for the developments presented in [5]. The main differences of our paper with [5] are: (i) we focus on signature schemes, (ii) the three main algorithms are modified to represent the new steps and (iii) communication now takes place in only one way due to the nature of digital signature protocols.

Finally, regarding the post-quantum signature scheme FALCON, as far as the authors know there are no formal specifications or analyses in the literature.

3. Preliminaries

The section briefly introduces the language used to specify our model, Maude. Moreover, a high-level view of the framework used to specify the signature schemes is explained. Finally, an explanation of the nature and general behaviour of signature schemes is given.

3.1. Maude

Maude [12, 11] is a high-level programming language and system implementing rewriting logic [29]. Rewriting logic is ideally suited to specify and execute computational systems in a simple and natural way. Some examples are the works in Petri nets [30], process calculus [31], object-based systems [32], asynchronous hardware [33], a mobile ad hoc network protocol [34], cloud-based storage systems [35], web browsers [36], programming languages with threads [37], distributed control systems [38] and models of mammalian cell pathways [39, 40].

Rewriting logic has a sub-logic called membership equational logic. This sub-logic defines a system's deterministic parts using functional modules. In contrast, Maude system modules represent concurrent systems as conditional rewrite theories that model a nondeterministic system which may never terminate and where the notion of a computed value may be meaningless. In this concurrent system, the membership equational sub-theory defines the states of such a system as the elements of an algebraic data type, such as terms in an equivalence class associated with cryptography properties. We can call this aspect the static part of the specification. Instead, its dynamics, i.e., how states evolve, are described by the transition rules, which specify the possible local concurrent transitions of the system.

In the case of analysis, Maude provides a series of tools to analyze specified models. The most basic form of system analysis is illustrated by the `search` command in Maude. The command performs reachability analysis from an initial state to a target state, searching for states that violate the invariant. If the invariant fails to hold, it will do so for some finite sequence of transitions from the initial state, which will be uncovered by the `search` command above since all reachable states are explored in a breadth-first manner. If the invariant does hold, we may be lucky and have a finite state system, in which case the `search` command will report failure to find a violation of the invariant. However, the search will never terminate if an infinite number of states are reachable from the initial state. Moreover, under the assumption that the set of states reachable from an initial state is finite, Maude also supports explicit-state model checking verification of any properties in linear-time temporal logic (LTL) through its LTL model checker.

3.2. Framework

Our work is developed on the framework proposed from [5]. The framework's purpose is to ease the specification of Key Encapsulation Mechanism (KEMs) in a modular way, splitting the different components into modules. Moreover, with the specifications at hand, the framework allows the user to run analyses such as invariant analysis or model checking of properties. In [5], the authors present three case studies where Kyber, Classic McEliece and BIKE are modelled and analysed. These KEMs were candidates of the same round but under the Public-key Encryption and Key-establishment Algorithm section.

A general picture of the framework is given in Figure 1. There are different modules in the framework, depicted by the colours of the box, and inclusion relations between modules are represented by arrows. Functional modules are in blue, and system modules are in red. The data structures and main algorithms (KeyGen, Enc and Dec) are represented in functional modules through symbols and equations, e.g. `DATA-TYPES` module specifies the different data types and their properties the scheme handles. Functional module `MODEL-CONFIGURATION`

serves as a basis for the system module, defining the structure upon which the behaviour of the scheme will be specified later. The general configuration handles participants in the protocol, a network for message passing between the participants, and lists of contact symbols to be used as keys and values in the symbolic computations. Finally, everything blends in the system module KEM. There, the communication actions of honest participants and intruders are represented by transition rules between states of the general configuration. Thanks to the data and cryptography properties defined through an equational theory in the functional modules, symbolic simulations of the protocol can be performed.

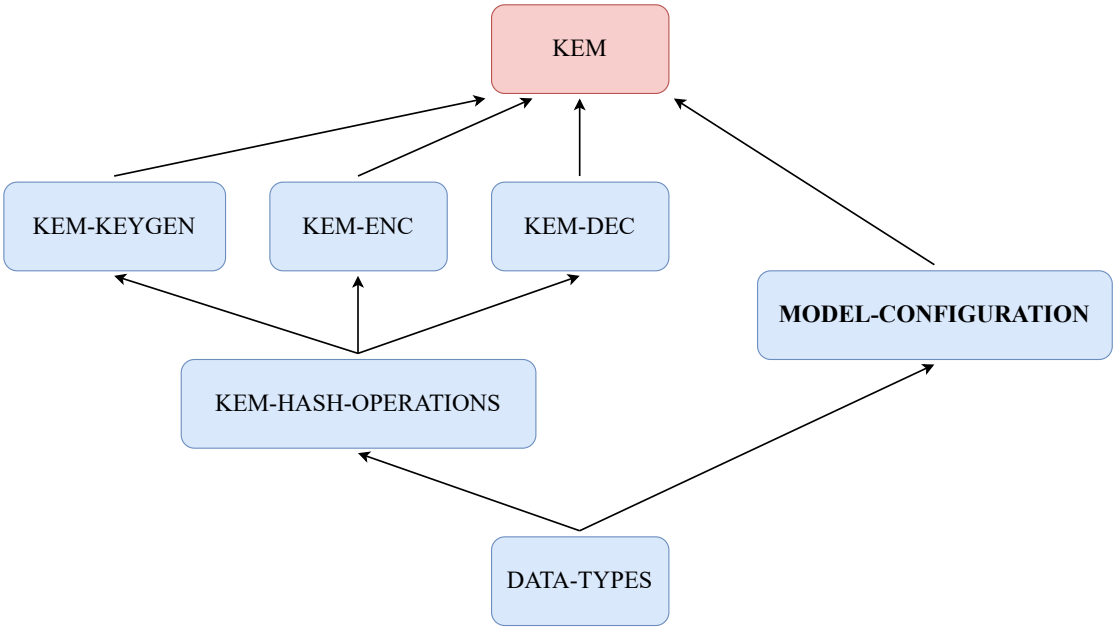


Figure 1: Overview of the framework, extracted from [5], used to specify and analyze post-quantum key encapsulation mechanisms in a modular way.

3.3. Signature schemes

A signature scheme is a mathematical scheme that uses asymmetric cryptography to verify the authenticity of digital assets, e.g. messages or documents. The asymmetric cryptography takes place in the form of a pair of keys, one public and one private. As an overview, the private key is used to sign the asset. Meanwhile, the public key, known by the other participant, is required to verify the validity of the generated signature. Signature schemes provide a mechanism for participants to verify that a message comes from someone they know, given that they know the public key of the sender.

Let's see an example of a general flow of a signature scheme. Figure 2 depicts a network diagram between two participants, Alice and Bob. The participants perform a series of actions depending on their role in the signature scheme. Let us suppose that Alice initiates the signature process since she wants to send a message, our digital asset, to Bob. To achieve this, Alice first needs to generate a pair of keys, a publicly known key and a secret key. After generating the

pair of keys, Alice uses the secret key over the message to generate a signature of the message. Moreover, since Alice wants to make sure Bob receives a valid version, she sends the signature along with the original message to Bob. Upon reception of the message and signature, Bob can validate the message's signature by using Alice's public key. If the signature proves valid, then Bob has strong guarantees that the message comes from Alice and thus knows the message is authentic and keeps its integrity.

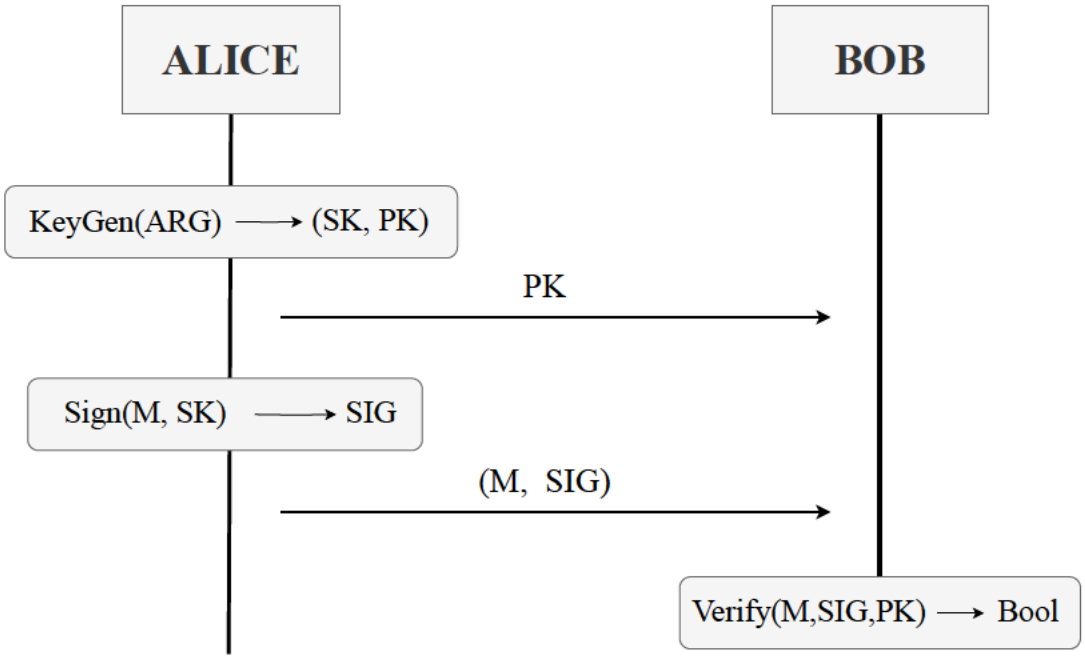


Figure 2: Network diagram of a signature scheme between Alice and Bob.

4. FALCON

Fast-Fourier Lattice-based Compact Signatures over NTRU [4] is a signature scheme based on lattices to sign and verify messages. The main components are a class of cryptography lattices, the NTRU class is chosen, and a trapdoor sampler, for which a new technique has been developed, the fast Fourier sampling. Concrete algorithms for each of the main three functions are presented in Figure 3, and will serve as guidelines to our Maude specifications.

The first algorithm, **KeyGen**, generates a pair of keys depending on two arguments: ϕ and q. These two arguments serve as seeds for the whole scheme and are required for the computation of polynomials f, g, F and G. These polynomials then form the components of a matrix used to compute what the authors call a FALCON tree. This tree, along with a normalized version of the matrix is what defines the secret key $sk = (\hat{B}, T)$. The public key is then given by the value $h = gf^{-1} \mod q$ in step 7 of the algorithm, where the first two polynomials are multiplied (one in inverse form) under the modulus of q, one of the arguments that serve as seed.

The algorithm to sign some digital assets, **Sign**, needs the digital asset in question and the private key. Note that, in principle, the algorithm would also receive a certain bound β, but we simplified the specification based on the assumptions given in Section 5.1. First, a random value r is uniformly sampled, to which the message to sign is concatenated and given to a hash operation to produce a value c. This hash value c is used along the components of the secret key to compute a pair of values s_1 and s_2. Finally, s_2 is compressed to serve as a component of the signature along with the random value r. Thus, the signature $sig = (r, s)$ is the pair of values of the uniformly sampled r and the compressed value s_2.

The verification algorithm, **Verify**, uses the sent asset along with its signature, together with the public key of the sender. Similar to the signing algorithm, a bound β is needed, but we omit it due to the assumptions given in Section 5.1. The verification procedure is as follows. The value c is recomputed using the received message m with the second component of the signature, i.e. the salt r. Then, values s_1 and s_2 are recomputed based on the recomputed value c. The value for s_2 is obtained through the decompression of the compressed value in s. Meanwhile, s_1 depends on the recomputed values of c and s_2, plus the public key, to be recomputed using the formula in step 3. If the recomputed values s_1 and s_2 match the original ones then the signature is proven to be valid for the received message m. Otherwise, the signature is rejected and the validity of the message can not be proven valid.

KeyGen : $(\phi, q) \rightarrow (sk, pk)$
1 : $f, g, F, G \leftarrow NTRUGen(\phi, q)$
2 : $B \leftarrow mat(f, g, F, G)$
3 : $\hat{B} \leftarrow FFT(B)$
4 : $G \leftarrow \hat{B} \times \hat{B}^*$
5 : $T \leftarrow ffLDL^*(G)$
6 : $sk \leftarrow (\hat{B}, T)$
7 : $h \leftarrow gf^{-1} \mod q$
8 : $pk \leftarrow h$

pk →

Sign : $(m, sk) \rightarrow sig$
1 : $r \leftarrow \{0,1\}^{320}$
2 : $c \leftarrow HashToPoint(r||m, q, n)$
3 : $t \leftarrow (FFT(c), FFT(0))\hat{B}^{-1}$
4 : $z \leftarrow ffSampling(t, T)$
5 : $s = (t - z)\hat{B}$
6 : $(s_1, s_2) \leftarrow invFFT(s)$
7 : $s \leftarrow Compress(s_2, 8 \cdot sbytelen - 328)$
8 : $sig = (r, s)$

(m, sig) →

Verify : $(m, sig, pk) \rightarrow Accept/Reject$
1 : $c \leftarrow HashToPoint(r||m, q, n)$
2 : $s_2 \leftarrow Decompress(s, 8 \cdot sbytelen - 328)$
3 : $s_1 \leftarrow c - s_2 h \mod q$

Figure 3: Falcon internal algorithms adapted from [4].

5. Formal specification with Maude

This section takes a closer look at the formal specification built in Maude. First, a series of assumptions over the model are made so we can abstract from computational requirements and focus on the symbolic behaviour, easing the specification. Then we explain how we used the existing framework to define the different rules, and how the equational theory represents the cryptography properties making the specification executable.

5.1. Assumptions

When symbolically modelling a protocol one can abstract himself from computational aspects to make the specifications simpler and easier to understand. Such abstractions come at a cost, making the model less representative of what really happens in the real world, but facilitating the reasoning of the intended behaviour. The assumptions we make for our symbolic specification over the computational model are:

- Any generated polynomials `f`, `g`, `F` and `G` always satisfy the equation $fG - gF = q \mod \phi$. This helps to encapsulate on the respective symbolic values any polynomial that will satisfy the equation.
- The signature and verification algorithms operate always in the bound given by their parameter β, thus it is not deemed necessary in the specification of these algorithms. The assumption allows us to avoid specifying the parameter in algorithms for signature generation and validation, easing the procedure. The main loss would be the symbolic model's fidelity over an actual implementation, but this can apply to most kinds of assumptions given their nature.
- The correctness of the encodings `s1` and `s2` in signature and validation algorithms. In other words, signature validation of a well-constructed signature for a valid message does not fail. This assumption allows us to focus on one of the outcomes of the validation algorithm, i.e. the case in which the signature is proven valid, skipping what would happen if the signature is rejected.
- Modulus operation over `q` is equal to modulus operation over `(q, `ϕ`)`. This is one of the most important assumptions we have made, since it will allow us to model the relation $s_1 + s_2 h = c \mod (\phi, q)$ to later obtain s_1 as the equivalent equality $c - s_2 h \mod q = s_1$.

5.2. Code specification

The framework from [5] serves as a perfect frame for specifying signature schemes. The main differences fall in (i) the participants' behaviour, since now messages go only one way through the communication channel, and (ii) the three main algorithms and their related modules. In the case of the communication, the rules used to send and receive messages are similar in both KEMs and DSAs. However, the main algorithms in digital signature schemes differ in content and application from those of KEMs, thus their modules hold new declarations to use in the redefined rules. We now dive into the details of each of the main algorithms of FALCON specified in Maude. Having in mind Figure 3 will help in the explanations.

5.2.1. KeyGen

Similar to KEMs, signature schemes require a key generation algorithm to provide a pair of keys. We have defined in module FALCON-KEYGEN, partially shown in Figure 4, the necessary symbolic values and operations for this algorithm. Specifically, we have defined operator constants phi and q to represent any possible value of ϕ and q respectively. The generator of polynomials *NTRUGen* is declared and defined through the equation to return a list of polynomials f, g, F and G, which are symbolic representations of any possible combination of them when receiving the constants phi and q as parameters. Furthermore, operations regarding Fast Fourier Transformation (FFT), the product between matrices and ffLDL have also been defined.

```
fmod FALCON-KEYGEN is
    ...
    --- Sample values for KeyGeneration
    op phi : -> Polynomial .
    op q : -> Nat .

    --- Sampler of values f, g, F and G
    op NTRUGen : Polynomial Nat -> List{Data} .
    ops f g F G : -> Polynomial .

    eq NTRUGen(phi, q) = (f g F G) .

    --- Matrix of a set of polynomials (this is to construct B)
    op mat : List{Data} -> Matrix .

    --- Fast Fourier Transformation
    op FFT : Matrix -> Polynomial .
    op FFT : Polynomial -> Polynomial .

    --- Matrix product
    op _x_ : Polynomial Polynomial -> Matrix .

    --- Fast Fourier LDL
    op ffLDL : Matrix -> Polynomial .
endfm
```

Figure 4: Functional module to declare and define values and operations to perform the key generation in FALCON.

The rule for key generation requires two samples from different groups of samples available as Figure 5 shows in the first line. Specifically, sample values for ϕ and q from the pools phis and qs are taken. These values serve as the basics for the construction of the public and secret keys. The conditions of the conditional rule KeyGen serve as representations for some of the internal steps in the algorithm. The first step is the generation of a list of polynomials L through function NTRUGen on the sampled values SAM1 and SAM2. Then, we collapse some of the specification

steps in the definition of the secret key SK. The secret key is defined in this second step as a pair of elements. The first element of the pair is the result of applying FFT over the resulting matrix from the elements of L, i.e. f, g, F and G. The second element of the pair is the product between ffLDL over the first element of the pair with that same element. Finally, the public key is defined as the product of the second element of L and the inverse of the first element of L, modulus the sampled value for q.

```
crl [KeyGen] :  {phis(SAM1, CONT1), qs(SAM2, CONT2), CONT3}
                < (ID1[emptyK]peer(ID2)) PS >
                net(MSGS)
                =>
                {phis(CONT1), qs(CONT2), CONT3}
                < (ID1[publicKey(ID1,PK) ; secretKey(ID1,SK)]
                   qI(ID1,SAM2), phiI(ID1,SAM1), peer(ID2)) PS >
                net(MSGS)
                if  L := NTRUGen(SAM1,SAM2) /\
                    SK := ([FFT(mat(L)),ffLDL(FFT(mat(L)) x FFT(mat(L)))]) /\
                    PK := (elem(2,L) p* inv(elem(1,L))) mod SAM2 /\
                    ID1 =/= ID2 .
```

Figure 5: Conditional rule for the specification of a participant applying the KeyGen algorithm.

5.2.2. Sign

Signature schemes require operations to sign the digital asset, in this case, a message. To this end, we defined a functional module named FALCON-SIGN, depicted in Figure 6. Similar to module FALCON-KEYGEN, the module stores the declarations of sample values necessary for the signature, as well as operations. Constant values are declared to represent the salt r and message m. Furthermore, operators s1 and s2 symbolically represent any value for s_1 and s_2 respectively. Regarding operations, the sampling procedure ffSampling and the inverse Fast Fourier Transformation invFFT are declared, plus the second one is also defined to return the pair of values s1 and s2. Finally, an equation representing the property $s1 + s2h = c \mod q$ is given. This will prove to be very useful when trying to validate the signature in the **Verify** algorithm.

Figure 7 depicts the rule that models the behaviour of a participant applying the Sign algorithm. The initial requirements for the rule to apply over a participant are: (i) there is some message to be created, (ii) there is a sample value for r available and (iii) the participant has a secret key and a value q obtained from the key generation algorithm. Then, the conditions of the rule specify how the components are defined following the specification of FALCON. The first step is to create a hash value c from the message STR and the salt r. In the next two steps, a preimage of c is stored in t and is given as a parameter to ffSampling along the secret key. The result is then put in a formula, following the pattern on the right-hand side of the equation for invFFT in Figure 6. This allows the final step to return the pair of short polynomials s1 and

```
fmod FALCON-SIGN is
    ...
    --- Sample values for r
    op r : -> Nat .

    --- Sample values for messages
    op m : -> String .

    --- Fast Fourier Sampling
    op ffSampling : Polynomial Polynomial -> Polynomial .

    --- Constant values to represent an instance of s_1 and s_2
    ops s1 s2 : -> Polynomial .

    --- Inverse Fast Fourier Transformation
    op invFFT : Polynomial -> Pair .
    eq invFFT((P1:Polynomial p- P2:Polynomial) p* FFT(M1:Matrix)) = ([s1,s2]) .

    --- Equation specifiyng equality "s1 + s2h = c mod (q)"
    eq (P1:Polynomial p- (s2 p* P2:Polynomial)) mod N:Nat = s1 .
endfm
```

Figure 6: Functional module to declare and define values and operations to sign a message in FALCON.

s2. Finally, a signature associated with STR is placed in the contents of the participant. This signature is composed of the sampled random salt r and the compressed value of *s2*. Take note that the second component is compressed with a value called SBYTELEN. This symbolic constant is defined to represent the value of the byte length of s.

```
crl [Sign] :    {ms(str(STR), CONT1), rs(SAM1, CONT2), CONT3}
                < (ID1[secretKey(ID1,SK) ; KS1]qI(ID1,Q), CONT4) PS >
                net(MSGS)
                =>
                {ms(CONT1), rs(CONT2), CONT3}
                < (ID1[KS1]sig(STR,SAM1,Compress(second(Ss),SBYTELEN)),
                          mI(ID1,STR), qI(ID1,Q), CONT4) PS >
                net(MSGS)
                if  c := HashToPoint(SAM1 || STR, Q, n) /\
                    t := [FFT(c), FFT(0)] p* inv(first(SK)) /\
                    z := ffSampling(t, second(SK)) /\
                    s := (t p- z) p* first(SK) /\
                    Ss := invFFT(s) .
```

Figure 7: Conditional rule for the specification of a participant applying the Sign algorithm.

5.2.3. Verify

The last algorithm in the scheme, Verify, is depicted with the conditional rule of the same name in Figure 8. The participant must be peers with another one and have received both a public key and a message along with its corresponding signature from that peer. Now, similar to the Signing algorithm a hash value `c` of the message is computed. As the condition shows, the random value `R` is obtained from the first component of the signature pair. Then, the recomputation of values `s2` and `s1` takes place. On the one hand, obtaining the value of `s2` is straightforward since `P = Compress(s2, SBYTELEN)`. Through the equation `Decompress(Compress(S,N),N) = S`, where `S` is some polynomial value and `N` is some numeric value, we obtain *s2*. On the other hand, the value of `s1` requires an equational theory based on a relation between values. The specific relation is given by the equality $s_1 + s_2 h = c \mod (\phi, q)$ available in Algorithm 10 in [4], which we translated into the equivalent equation $c - s_2 h \mod (\phi, q) = s_1$ by leaving s_1 on the right side alone. This equation makes explicit the relation between the hash c, the public key h and the pair of values s_1 and s_2. The last condition models the check that the computed values must be equal to those used during the signing, thus verifying the signature of the message.

```
crl [Verify] :  {CONT1}
                < (ID1[publicKey(ID2, p mod Q) ; KS1]
                   mI(ID2,STR), sig(STR,R,P), peer(ID2), CONT2) PS >net(MSGS)
                =>
                {CONT1}
                < (ID1[KS1]mI(ID2,STR), peer(none), CONT2) PS >
                net(MSGS)
                if  c := HashToPoint(R || STR, Q, n) /\
                    S2 := Decompress(P, SBYTELEN) /\
                    h := p mod Q /\
                    S1 := (c p- (S2 p* h)) mod Q /\
                    (S1 == s1) /\ (S2 == s2) .
```

Figure 8: Conditional rule for the specification of a participant applying the Verify algorithm.

5.2.4. Execution

To prove termination of our formal model we run the `rewrite` command to see if the initial state ends in a state where two honest participants, Alice and Bob, apply the specified rules. Figure 9 shows the result of applying the said command over initial state `init1`, which is defined as `phis(phi), qs(q), ms(str(m)), rs(r)< (Alice[emptyK]peer(Bob)) (Eve[emptyK]peer(none)) (Bob[emptyK]peer(Alice)) >net(emptyM)`, where from left to right it defines a configuration with: (1) a set of samples, each with one element available, (2) a set of participants with no keys and two of them set to begin the protocol between them, and (3) an empty network of messages, i.e. there is no previous history of message exchanges. When applying the command we get a new configuration where Alice and Bob finished the communication between them, thus they are no longer peers and Bob has the message Alice

wanted to send him. With it we have proven termination, a liveness property, of the model. We do not prove any other properties, and leave them for future work.

```
                    \||||||||||||||||||/
                  --- Welcome to Maude ---
                    /||||||||||||||||||\
           Maude 3.3.1 built: Apr 13 2023 16:09:10
            Copyright 1997-2023 SRI International
                  Fri Oct 20 12:49:40 2023
==========================================
rewrite in FALCON : init1 .
rewrites: 30 in 0ms cpu (0ms real) (240000 rewrites/second)
result [GlobalState]:
{phis(emptyC),qs(emptyC),ms(emptyC),rs(emptyC)}
<
(Alice[emptyK]peer(none),phiI(Alice, phi),qI(Alice, q))
(Eve[emptyK]peer(none))
Bob[emptyK]peer(none),mI(Alice, m)
>net(msg{(Alice,Bob)[received](g p* inv(f)) mod q}
       msg{(Alice,Bob)[received]str(m),[r,Compress(s2, SBYTELEN)]})
```

Figure 9: Execution of the rewrite command over an initial state `init1` to check if the symbolic specification represents an execution of the signature scheme FALCON.

6. Conclusion and future work

We provide a first approach on the formal specification of post-quantum signature schemes. Specifically we adapted the framework from [5] to serve as a tool for the specification of signature schemes. The post-quantum signature scheme FALCON served as our case study to show the feasibility of the Maude language to represent the behaviour of post-quantum signature schemes along with their cryptography properties.

This work serves as a first step towards the formal verification of signature schemes. Thus, as future work we plan to use explicit state model checking for the verification of invariants and LTL Model Checking to check if any interesting property holds in the specified symbolic model.

References

[1] R. L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (1978) 120–126.

[2] W. J. Caelli, E. P. Dawson, S. A. Rea, Pki, elliptic curve cryptography, and digital signatures, Computers & Security 18 (1999) 47–66.

[3] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM review 41 (1999) 303–332.

[4] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, Z. Zhang, Falcon: Fast-fourier lattice-based compact signatures over ntru (2020).
[5] V. García, S. Escobar, K. Ogata, S. Akleylek, A. Otmani, Modelling and verification of post-quantum key encapsulation mechanisms using maude, PeerJ Computer Science 9 (2023) e1547.
[6] B. Blanchet, Security protocol verification: Symbolic and computational models, in: International Conference on Principles of Security and Trust, Springer, 2012, pp. 3–29.
[7] V. Cortier, S. Kremer, B. Warinschi, A survey of symbolic methods in computational analysis of cryptographic systems, Journal of Automated Reasoning 46 (2011) 225–259.
[8] M. Abadi, P. Rogaway, Reconciling two views of cryptography (the computational soundness of formal encryption), Journal of cryptology 15 (2002) 103–127.
[9] S. Escobar, C. Meadows, J. Meseguer, Maude-NPA: Cryptographic protocol analysis modulo equational properties, in: Foundations of Security Analysis and Design V, Springer, 2009, pp. 1–50.
[10] S. Escobar, C. Meadows, J. Meseguer, A rewriting-based inference system for the nrl protocol analyzer and its meta-logical properties, Theoretical Computer Science 367 (2006) 162–202.
[11] M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln, N. Martí-Oliet, J. Meseguer, R. Rubio, C. Talcott, Maude Manual (version 3.3.1), Technical Report, SRI International, 2023. URL: http://maude.cs.illinois.edu.
[12] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. M. Oliet, J. Meseguer, C. Talcott, All About Maude - A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic, Lecture Notes in Computer Science, Springer, 2007. URL: http://dx.doi.org/http://dx.doi.org/10.1007/978-3-540-71999-1. doi:10.1007/978-3-540-71999-1.
[13] F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer, R. Rubio, C. Talcott, Programming and symbolic computation in maude, Journal of Logical and Algebraic Methods in Programming 110 (2020) 100497.
[14] B. Blanchet, B. Smyth, V. Cheval, M. Sylvestre, Proverif 2.00: automatic cryptographic protocol verifier, user manual and tutorial, Version from (2018) 05–16.
[15] B. Blanchet, V. Cheval, V. Cortier, Proverif with lemmas, induction, fast subsumption, and much more, in: 2022 IEEE Symposium on Security and Privacy (SP), IEEE, 2022, pp. 69–86.
[16] S. Meier, B. Schmidt, C. Cremers, D. Basin, The tamarin prover for the symbolic analysis of security protocols, in: International conference on computer aided verification, Springer, 2013, pp. 696–701.
[17] D. Basin, C. Cremers, J. Dreier, R. Sasse, Tamarin: verification of large-scale, real-world, cryptographic protocols, IEEE Security & Privacy 20 (2022) 24–32.
[18] C. J. Cremers, The scyther tool: Verification, falsification, and analysis of security protocols, in: International conference on computer aided verification, Springer, 2008, pp. 414–418.
[19] J. Ramsdell, J. Guttman, CPSA4: A cryptographic protocol shapes analyzer, https://github.com/mitre/cpsaexp, 2018.
[20] I. Gazeau, S. Kremer, Automated analysis of equivalence properties for security protocols using else branches, in: Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings,

Part II 22, Springer, 2017, pp. 1–20.
[21] V. Cheval, S. Kremer, I. Rakotonirina, The DEEPSEC prover, in: International Conference on Computer Aided Verification, Springer, 2018, pp. 28–36.
[22] S.-L. Gazdag, S. Grundner-Culemann, T. Guggemos, T. Heider, D. Loebenberger, A formal analysis of IKEv2's post-quantum extension, in: Annual Computer Security Applications Conference, 2021, pp. 91–105.
[23] C. Jacomme, E. Klein, S. Kremer, M. Racouchot, A comprehensive, formal and automated analysis of the edhoc protocol, in: USENIX Security'23-32nd USENIX Security Symposium, 2023.
[24] A. Hülsing, K.-C. Ning, P. Schwabe, F. Weber, P. R. Zimmermann, Post-quantum wireguard, in: 2021 IEEE Symposium on Security and Privacy (SP), IEEE, 2021, pp. 304–321.
[25] D. D. Tran, C. M. Do, S. Escobar, K. Ogata, Hybrid post-quantum tls formal specification in maude-npa-toward its security analysis?, Proceedings http://ceur-ws.org ISSN 1613 (2022) 0073.
[26] D. D. Tran, C. M. Do, S. Escobar, K. Ogata, Hybrid post-quantum transport layer security formal analysis in maude-npa and its parallel version, PeerJ Computer Science 9 (2023) e1556.
[27] D. D. Tran, K. Ogata, S. Escobar, S. Akleylek, A. Otmani, Formal specification and model checking of lattice-based key encapsulation mechanisms in Maude, in: Rewriting Logic and its Applications 14th International Workshop, WRLA 2022, 2022, p. 26.
[28] D. D. Tran, K. Ogata, S. Escobar, S. Akleylek, A. Otmani, Formal specification and model checking of saber lattice-based key encapsulation mechanism in Maude, in: Proceedings of the 34th International Conference on Software Engineering and Knowledge Engineering, 2022.
[29] J. Meseguer, Conditional rewriting logic as a united model of concurrency, Theor. Comput. Sci. 96 (1992) 73–155. URL: https://doi.org/10.1016/0304-3975(92)90182-F. doi:10.1016/0304-3975(92)90182-F.
[30] M.-O. Stehr, J. Meseguer, P. C. Ölveczky, Rewriting logic as a unifying framework for petri nets, in: Unifying Petri Nets, Springer, 2001, pp. 250–303.
[31] N. Martí-Oliet, J. A. Verdejo-López, Implementing CCS in Maude, in: Actas de las VIII Jornadas de Concurrencia: Cuenca, 14 a 16 de junio de 2000, Universidad de Castilla-La Mancha, 2000, pp. 81–96.
[32] J. Meseguer, A logical theory of concurrent objects and its realization in the Maude language, in: G. Agha, P. Wegner, A. Yonezawa (Eds.), Research Directions in Concurrent Object-Oriented Programming, MIT Press, 1993, pp. 314–390.
[33] M. Katelman, S. Keller, J. Meseguer, Rewriting semantics of production rule sets, Journal of Logic and Algebraic Programming 81 (2012) 929–956.
[34] S. Liu, P. C. Ölveczky, J. Meseguer, Modeling and analyzing mobile ad hoc networks in Real-Time Maude, Journal of Logical and Algebraic Methods in Programming (2015).
[35] R. Bobba, J. Grov, I. Gupta, S. Liu, J. Meseguer, P. Ölveczky, S. Skeirik, Design, Formal Modeling, and Validation of Cloud Storage Systems using Maude, in: R. H. Campbell, C. A. Kamhoua, K. A. Kwiat (Eds.), Assured Cloud Computing, J. Wiley, 2018, pp. 10–48. URL: http://hdl.handle.net/2142/96274.
[36] S. Chen, J. Meseguer, R. Sasse, H. J. Wang, Y.-M. Wang, A systematic approach to uncover

security flaws in gui logic, in: 2007 IEEE Symposium on Security and Privacy (SP'07), IEEE, 2007, pp. 71–85.

[37] J. Meseguer, G. Roşu, The rewriting logic semantics project, Theoretical Computer Science 373 (2007) 213–237.

[38] K. Bae, J. Meseguer, P. C. Ölveczky, Formal patterns for multirate distributed real-time systems, Science of Computer Programming 91 (2014) 3–44.

[39] S. Eker, M. Knapp, K. Laderoute, P. Lincoln, J. Meseguer, K. Sonmez, Pathway logic: Symbolic analysis of biological signaling, in: Biocomputing 2002, World Scientific, 2001, pp. 400–412.

[40] C. Talcott, S. Eker, M. Knapp, P. Lincoln, K. Laderoute, Pathway logic modeling of protein functional domains in signal transduction, in: Biocomputing 2004, World Scientific, 2003, pp. 568–580.

A formal analysis of OpenPGP's post-quantum public-key algorithm extension

Duong Dinh Tran[1,*], Kazuhiro Ogata[1] and Santiago Escobar[2]

[1]Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan
[2]VRAIN, Universitat Politècnica de València, Valencia, Spain

Abstract

OpenPGP is an open standard widely used for encrypting and decrypting communication data. Because the security of the current public-key algorithms is dangered by the rapid advancement of quantum computers, a post-quantum extension of the OpenPGP protocol has been proposed in which traditional public-key primitives based on elliptic curve cryptography (ECC) are used in combination with CRYSTALS-Kyber and CRYSTALS-Dilithium, two presumed quantum-resistant schemes. This paper presents a formal analysis of the OpenPGP's post-quantum extension with Maude, a formal specification language and a high-performance tool. We model the protocol in Maude as a state machine with the presence of two honest participants together with an active attacker who can intercept network connections and specifically, break the ECC-based public-key algorithms. The security properties under analysis are (1) *secrecy of messages* and (2) *authenticity of messages*. By using the Maude search functionality, we verify that the protocol enjoys (1) and (2) up to depths 13 and 11, respectively.

Keywords

OpenPGP, post-quantum, formal analysis, security verification

1. Introduction

OpenPGP [1, 2] is an open standard of the Pretty Good Privacy (PGP) security program, a popular system used for encrypting and decrypting communications, preventing unwanted individuals from eavesdropping. The most widely used application of the OpenPGP protocol is to allow individuals to exchange emails securely by encrypting and decrypting their contents. Besides, OpenPGP also supports protecting private texts, files, and even whole disk partitions. OpenPGP supports many public-key encryption and digital signature algorithms. Initially, the RSA algorithm was used as the public-key encryption scheme. To protect a message, the sender encrypts it with a random session key that is then encrypted by the receiver's RSA public key. Only the recipient can decrypt the session key by using the RSA private key and uses it to decrypt the encrypted message. Over time, to meet the security requirement, the RSA key size was increased, which made it lose efficiency. Elliptic Curve Cryptography (ECC), on the other hand, can provide the same security level as RSA with a smaller key size. The OpenPGP standard was then extended to support signing and key exchange based on ECC by RFC 6637 [3]

FAVPQC 2023: International Workshop on Formal Analysis and Verification of Post-Quantum Cryptographic Protocols, November 21, 2023, Brisbane, Australia.
*Corresponding author.
duongtd@jaist.ac.jp (D. D. Tran); ogata@jaist.ac.jp (K. Ogata); sescobar@upv.es (S. Escobar)

in 2012. Instead of using the recipient's RSA public key and private key to encrypt and decrypt the session key, the sender and recipient use a shared secret established by the Elliptic Curve Diffie-Hellman (ECDH) method to symmetrically encrypt the session key.

The security of the currently used public-key algorithms, including RSA, ECDH, and ECC-based digital signatures, is threatened by the rapid advancement of quantum computers recently. On a sufficiently large quantum computer, Shor's algorithm [4] allows attackers to break those public-key algorithms by efficiently computing the private key from the public key. Facing that threat, in 2017, The National Institute of Standards and Technology (NIST) started the Post-Quantum Cryptography Standardization Project [5], which aims to standardize a new class of public-key algorithms that are resistant to quantum computers as measures before practical quantum computers become operational. A post-quantum extension for the OpenPGP protocol [6] has been proposed and standardized by an Internet Engineering Task Force (IETF) working group. In this extension, a traditional ECC-based public-key encryption scheme is used in combination with CRYSTALS-Kyber [7, 8], a Key Encapsulation Mechanism (KEM), and an ECC-based digital signature is used in combination with CRYSTALS-Dilithium [9]. CRYSTALS-Kyber and CRYSTALS-Dilithium base their security on the hardness of the *learning with errors* problem, which is presumed to be hard even with quantum computers. They are two of the first group of winners in the standardization project as NIST announced in July 2022[1]. Let us shortly refer to them as Kyber and Dilithium. The OpenPGP's post-quantum extension is referred to as PQ OpenPGP in this paper.

In this paper, we present a formal analysis of PQ OpenPGP. We use Maude [10], a formal specification language and a high-performance tool to model the protocol as a state machine. We employ the equipped invariant model checker in Maude (the **`search`** command) to check two properties: (1) *secrecy of messages* and (2) *authenticity of messages*. The former guarantees that messages are securely exchanged, that is, no one other than the sender and the recipient can learn the message content. The latter states that if Bob successfully decrypts an encrypted message apparently sent from Alice and verifies the signature, obtaining a raw message M, then Alice indeed sent M. To model the presence of an attacker, we rely on the standard Dolev-Yao model [11]. There exists a generic attacker who can control all connections in the network. Particularly, the attacker can intercept any message sent in the network to collect information from such a message, use all cryptographic primitives to manipulate the available information, and pretend some individual to send a faking message to another individual. Besides, our specification also allows the attacker to break the security of the ECDH method and the ECC-based digital signature schemes, that is, the attacker can derive the associated private key from a given public key.

The Maude specification of the protocol is publicly available on the webpage[2]. The remainder of this paper is organized as follows. Some closely related work is mentioned in Section 2. Next, in Section 3, we briefly introduce some preliminaries, which are necessary for the rest of the paper. OpenPGP and its post-quantum extension are described in Section 4. Then, Section 5 and Section 6 present how to model the protocol in Maude and the formal analysis. Finally, Section 7 summarizes our study.

[1] https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
[2] https://github.com/duongtd23/pq-openpgp-analysis

2. Related work

There are several case studies on applications of formal methods to security analysis of post-quantum cryptographic protocols. S. Gazdag et al. [12] have conducted a security analysis of a post-quantum version of the Internet Key Exchange version 2 (IKEv2). IKEv2 is used to set up a security association (SA) in the Internet Protocol Security (IPsec) protocol [13], the most popular technology for setting up Virtual Private Networks (VPNs). They showed that in a quantum setting, i.e., an attacker has access to quantum computers, the original IKEv2 is insecure, which is similar to what we showed that Eve can learn the ECDH shared secret. They analyzed an IKEv2's post-quantum extension proposed in [14, 15], showing that the improved protocol enjoys all desired security properties. The formal method tool Tamarin [16] was used for the verification, where Tamarin is well known as one of the most useful tools for formal analysis of cryptographic protocols. The Tamarin specification of the protocol is essentially a state machine in which each state is an AC-collection of *facts* and transitions between states, which model the protocol execution, the behavior of honest participants, and capabilities of the attacker, are specified by Tamarin *rules*. Tamarin *facts* and *rules* are closely similar to observable components (i.e, name-value pairs) and Maude rewrite rules, respectively. To check a property, the tool performs an exhaustive, symbolic search based on constraint solving until either a satisfying trace is found or no more rewrite rules can be applied. Because symbolic search is executed, Tamarin can give security proof with respect to an unbounded number of sessions and protocol participants. Meanwhile, with our analysis reported in this paper, the number of participants must be fixed.

A. Hülsing et al. [17] have proposed a WireGuard's post-quantum extension and presented a formal verification of the desired security properties inherited from WireGuard [18], a VPN protocol focusing on simplicity, fast speed, and high performance. The tool Tamarin [16] was also used for the formal verification. They modeled the primitives, messages, etc., used in the protocol as function symbols and terms, which is similar to our approach with Maude. In addition, the paper also presented a computational security proof, which gave stronger security guarantees than the symbolic proof with Tamarin since probability and complexity are taken into account and fewer idealizing assumptions are made. However, more security properties are verified in the verification with Tamarin, and more importantly, it is computer-verified.

There are some existing case studies on analyzing cryptographic protocols and primitives also by using Maude [19, 20]. In this paper, to verify the security properties, we use only the Maude search command (or invariant model checker). Whereas, liveness and fairness properties were taken into account in the analysis experiments reported in [20]. They employed the LTL model checker in Maude. On the other hand, they modeled the protocol under analysis at a higher level than us, that is, some more simplifications are made in their model.

The Amazon Web Services has proposed a quantum-resistant version of the Transport Layer Security 1.2 protocol [21], called the Hybrid Post-Quantum TLS [22], and its formal analysis has been studied in [23]. The analysis was conducted by using Maude-NPA [24] and Par-Maude-NPA [25], a parallel version of Maude-NPA. Maude-NPA is a cryptographic protocol analysis tool implemented in Maude, based on narrowing & rewriting logic [26]. They used *strands* [27] to model the execution of the Hybrid Post-Quantum TLS protocol as well as the capabilities of a Dolev-Yao attacker. Two properties are verified: (1) secrecy property of the post-quantum

key encapsulation mechanism's shared secret, and (2) authentication property. To check each property, a Maude-NPA attack state is defined, representing the insecure state that violates the property. For the analysis, different from Maude's forward search, Maude-NPA performs a backward search from the attack state to check whether it is reachable from an initial state. The backward reachability analysis is symbolic, and hence, Maude-NPA can provide security proof with respect to an unbounded number of sessions and participants, which is an advantage over our analysis approach with Maude. Because of the very large state space generated, similar to our results reported in this paper, they can only guarantee that the protocol enjoys (1) and (2) up to depths 12 and 18, respectively, after about 72 days.

3. Preliminaries

This section gives the definition of key encapsulation mechanisms followed by a brief description of the Maude language.

Key Encapsulation Mechanism

According to the NIST standardization project, key exchange algorithms are formulated as Key Encapsulation Mechanisms (KEMs). The following is the definition of KEMs.

Definition 1. A key encapsulation mechanism is a tuple of algorithms (`KeyGen`, `Encaps`, `Decaps`) along with a finite key space $\mathcal{K}$:

- `KeyGen`() $\rightarrow (pk, sk)$: A probabilistic *key generation* algorithm that outputs a public key pk and a private key sk.
- `Encaps`(pk) $\rightarrow (c, k)$: A probabilistic *encapsulation* algorithm that takes as input a public key pk, and outputs a ciphertext (or encapsulation) c and a shared secret $k \in \mathcal{K}$.
- `Decaps`(c, sk) $\rightarrow k$: A deterministic *decapsulation* algorithm that takes as inputs a ciphertext c and a private key sk, and outputs a shared secret $k \in \mathcal{K}$.

Maude

Maude is a declarative language and high-performance tool that focuses on simplicity, expressiveness, and performance to support the formal specification and analysis of concurrent programs/systems in rewriting logic. In Maude, the most basic building block is *modules*. There are two kinds of modules in Maude: functional modules and system modules. A functional modules is in the form of **`fmod`** $\mathcal{M}$ **`is`** (Σ, E) **`endfm`**, where $\mathcal{M}$ is the module name, Σ is an order-sorted signature and E is an equation set. Σ may contain declarations of sorts (or types), subsorts, operators (or function symbols), and variables. A system module is in the form of **`mod`** $\mathcal{R}$ **`is`** (Σ, E, R) **`endm`**, where R is a set of rewrite rules. Using rewrite rules, we can specify state transitions of a state machine. A rewrite rule starts with the keyword **`rl`**, followed by a label enclosed with square brackets and a colon, two patterns connected with =>, and ends with a full stop. A conditional one starts with the keyword **`crl`** and has a condition following the keyword **`if`** before a full stop. The following is a form of a conditional rewrite rule:

$$\texttt{crl}\ [lb]\ :\ l\ \texttt{=>}\ r\ \texttt{if}\ \ldots\ \texttt{/\ }\ c_i\ \texttt{/\ }\ \ldots\ .$$

where lb is a label and c_i is part of the condition, which may be an equation $lc_i = rc_i$. The negation of $lc_i = rc_i$ could be written as $(lc_i\ \texttt{=/=}\ rc_i) = \texttt{true}$, where `= true` could be omitted. If the condition $\ldots$ `/\` c_i `/\` $\ldots$ holds under some substitution σ, $\sigma(l)$ can be replaced with $\sigma(r)$.

Maude provides the search command that can find a state reachable from t such that the state matches p and satisfies condition(s) c:

$$\texttt{search}\ \texttt{[n,m]}\ \texttt{in}\ \texttt{MOD}\ :\ t\ \texttt{=>*}\ p\ \texttt{such that}\ c\ .$$

where `MOD` is the name of the module specifying the state machine, and `n` and `m` are optional arguments stating a bound on the number of desired solutions and the maximum depth of the search, respectively. `n` typically is 1 and t typically represents an initial state of the state machine.

4. OpenPGP's post-quantum extension

We describe the OpenPGP protocol followed by its post-quantum extension.

OpenPGP and ECC in OpenPGP

The process that A (the sender) sends a message M to B (the recipient) is shown in Figure 1, which consists of the following sequence of steps:

1. The sender hashes the message (using the hash function H), and then signs the digest with his/her private key (sk_A).
2. The sender randomly selects a key value k, which is served as a *session key* to encrypt the message. Note that this session key is used for this message only, i.e., different messages use different session keys. For the analysis, we suppose that the session key is unique and non-guessable.
3. The sender asymmetrically encrypts (aenc) the session key with the recipient's public key (pk_B), obtaining C_1.
4. The sender concatenates the message's signature and the message, and symmetrically encrypts (senc) it under the session key k, obtaining C_2 ($||$ denotes the concatenation). The concatenation of C_1 and C_2 are outputted as the result.

The process shown in Figure 1 provides confidentiality and integrity to the message being sent. Depending on the sender's desire, it may not be desirable to have both security services. For instance, when integrity is not desirable, the sender will omit the signing part and proceed to encrypt the message only. In this study, however, we always consider both confidentiality and integrity to be desirable.

Figure 2 depicts how the recipient decrypts the received ciphertext and validates it, which consists of the following step sequence:

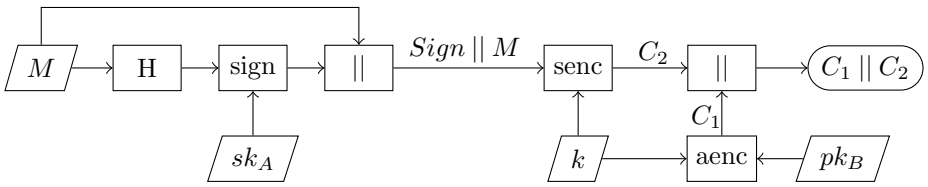


Figure 1: OpenPGP: Sending a message M

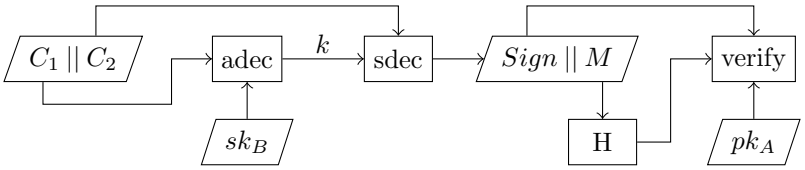


Figure 2: OpenPGP: Receiving an encrypted message

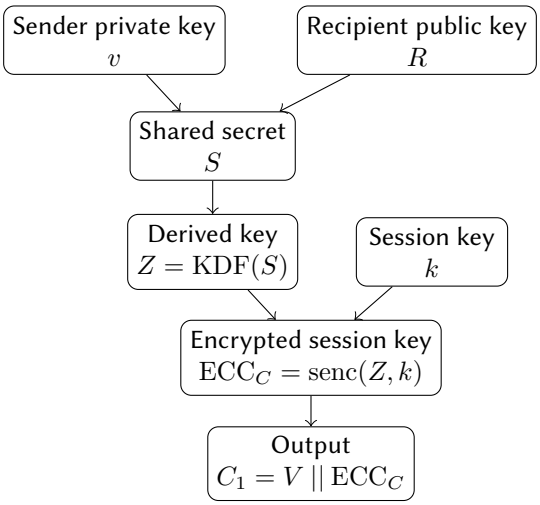


Figure 3: Session key encryption with ECDH

1. The recipient decrypts the first part of the ciphertext (C_1) with his/her private key (sk_B) to recover the session key k.
2. The recipient uses the session key to decrypt the second part of the ciphertext (C_2), obtaining a message and a signature.
3. The recipient hashes the message and uses the sender public key (pk_A) to validate the integrity of the message received.

When the ECDH key exchange method is used, instead of using the recipient's public key and private key to encrypt and decrypt the session key as shown in Figure 1 and Figure 2, a so-called *derived key* is computed from the shared secret and that derived key is used to encrypt

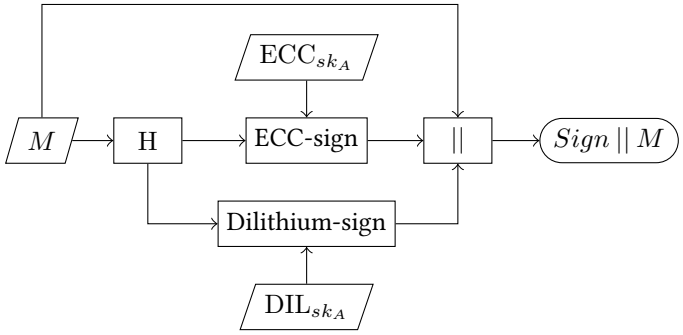


Figure 4: Composite ECC-based signature scheme and CRYSTALS-Dilithium

the session key. The precise step sequence to encrypt the session key with ECDH is given in Figure 3:

1. The sender first generates an ECDH ephemeral private/public key pair $\{v, V\}$.
2. The ECDH shared secret S is computed from v and the recipient public key R.
3. The sender then uses the key derivation function (KDF) to compute the derived key Z from the shared secret.
4. The session key k is encrypted under the derived key.
5. Finally, the sender concatenates his/her ephemeral public key and the encrypted session key, outputting the result.

Upon receiving the message, the recipient computes the shared secret from his/her ECDH private key (the associated private key of R) and the sender public key V. The recipient then derives Z, uses it to decrypt the session key, and uses the session key to decrypt the message as well as validate the signature.

OpenPGP's post-quantum extension

PQ OpenPGP makes use of two pairs of combinations: (1) an ECC-based signature algorithm in combination with CRYSTALS-Dilithium, and (2) ECDH in combination with CRYSTALS-Kyber. Figure 4 illustrates the composite ECC-based and Dilithium signature schemes. Given a message M, the sender hashes it, and signs the message digest with both ECC-based's and Dilithium's sign functions. Note that ECC$_{sk_A}$ and DIL$_{sk_A}$ denote the ECC-based and Dilithium signing private key of the sender, respectively. When decrypting the message, the recipient must validate both signatures.

The Internet-Draft [6] also defines another option in which SPHINCS+ [28] is used as a standalone public-key signature scheme. That option, however, is omitted in this paper.

Figure 5 depicts the composite algorithm with ECDH and Kyber to encrypt a session key k, which consists of the following sequence of steps:

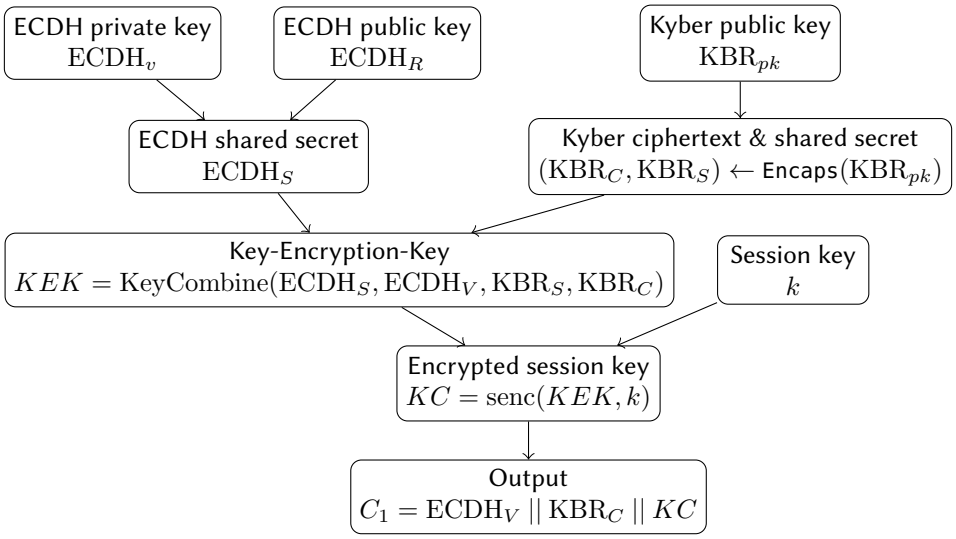


Figure 5: Composite ECDH and CRYSTALS-Kyber

1. The sender first generates an ECDH ephemeral private/public key pair $\{\mathrm{ECDH}_v, \mathrm{ECDH}_V\}$.
2. The ECDH shared secret ECDH_S is computed from the sender's private key and the recipient's public key ECDH_R.
3. Taking the Kyber public key of the recipient as input, the sender performs Kyber `Encaps` to generate a ciphertext KBR_C and a shared secret KBR_S. Note that because `Encaps` is probabilistic, each time it is invoked, different ciphertext and shared secret will be generated.
4. The sender then uses a so-called function $\mathrm{KeyCombine}$ to compute the so-called Key-Encryption-Key KEK from the two shared secrets, the sender's ECDH public key, and the Kyber ciphertext.
5. The session key k is encrypted under the Key-Encryption-Key.
6. Finally, the sender concatenates his/her ECDH public key, the Kyber ciphertext, and the encrypted session key, outputting the result.

Upon receiving the message, the recipient computes: (1) the ECDH shared secret from his/her private key (the associated private key of R) and the sender public key V; and (2) the Kyber shared secret by performing `Decaps`, taking his/her private key (the associated private key of KBR_{pk}) and the received Kyber ciphertext as inputs. The recipient then derives Key-Encryption-Key, uses it to decrypt the session key, and uses the session key to decrypt the message as well as validate the two signatures.

For the analysis, we suppose that the binding between a public key and its owner identifier is always correctly done. It means that the attacker is unable to claim their public key is a public

key of some honest participant. To achieve this purpose, different methods are used together in OpenPGP, such as, public-key fingerprints, certificates, and web of trust. Web of trust is a decentralized trust model, in contrast to the centralized trust model that relies on a certificate authority, allowing anyone to sign a partner public key to certify that that public key truly belongs to that person. Everyone can choose their trusted introducers, from which they trust all certificates signed by such introducers. In fact, however, no completely satisfactory solution (including the web of trust model) is known to the issue of binding between a public key and its owner.

5. Modeling the protocol

This section presents how to model PQ OpenPGP in Maude [10], a formal specification language and a high-performance tool that can be used to analyze a wide range of systems/protocols.

5.1. Modeling ECDH and CRYSTALS-Kyber

ECDH is modeled in the following Maude functional module:

```
fmod ECDH is
  sorts EdPubKey EdPriKey EdShareS .
      derivation of the associated public key from a private key
  op pk  : EdPriKey            -> EdPubKey .
      computation of the shared secret from a received public key and a private key
  op ss  : EdPubKey EdPriKey   -> EdShareS .
      constructor of a shared secret is a private key pair
  op  |  : EdPriKey EdPriKey   -> EdShareS [comm] .
  vars SK SK2 : EdPriKey .            declarations of variables
  eq ss(pk(SK), SK2) = SK | SK2 .     an equation
endfm
```

This module first introduces three sorts (or types) `EdPubKey`, `EdPriKey`, and `EdShareS` representing ECDH public keys, private keys, and shared secrets, respectively. We then declare three operators (or function symbols), where the comments followed by give their corresponding explanations for what they stand for. For instance, the first operator takes a private key as input and returns its associated public key as output. The equational attribute `comm` attached to the last operator denotes that it is commutative, namely, $(sk_1 \mid sk_2)$ and $(sk_2 \mid sk_1)$ are identical with all private keys sk_1 and sk_2. The equation defines the semantic of the last operator.

We model Kyber based on the general definition of KEMs, i.e., Definition 1. Another module is introduced, which contains the following declarations and definitions:

```
sorts KbPubKey KbPriKey KbShareS KbCipher .
    KeyGen is a probabilistic algorithm,
    so keygen takes a private key as input and returns the public key
op keygen  : KbPriKey          -> KbPubKey .
    similarly, Encaps is probabilistic, so an argument of KbPriKey is added
op encapsC : KbPubKey KbPriKey -> KbCipher .       Encaps: returns ciphertext
op encapsK : KbPubKey KbPriKey -> KbShareS .       Encaps: returns shared secret
op decaps  : KbCipher KbPriKey -> KbShareS .       Decaps
```

```
    constructor of a shared secret is a private key pair
op  &      : KbPriKey KbPriKey -> KbShareS .
vars SK SK2 : KbPriKey .
eq encapsK(keygen(SK), SK2) = (SK & SK2) .
eq decaps(encapsC(keygen(SK), SK2), SK) = (SK & SK2) .
```

We introduce four sorts `KbPubKey`, `KbPriKey`, `KbShareS`, and `KbCipher`, representing public keys, private keys, shared secrets, and ciphertexts (or encapsulations), respectively. As indicated in Definition 1, `KeyGen` and `Encaps` are two probabilistic algorithms. Thus, to specify them as deterministic procedures in Maude, an argument of the sort `KbPriKey` is added as the input argument. With `Encaps`, we introduce two separate operators `encapsC` and `encapsK`, respectively returning the ciphertext and the shared secret. The first equation defines the semantic of the operator `&`. Given a ciphertext and a private key, the second equation states that `Decaps` properly outputs the shared secret only if the given ciphertext encapsulates the public key that is the associated public key of the given private key.

5.2. Modeling cryptographic primitives

We define the generic sort `Data` as the supersort of all sorts mentioned before, such as `EdPubKey` and `KbPubKey`. `||` is the concatenation operator, which forms terms of the sort `DataL`.

```
sorts Data DataL .
subsorts EdPubKey EdPriKey KbPubKey KbPriKey ... < Data .      some other sorts in ...
subsort Data < DataL .
op nilDL :             -> DataL [ctor] .
op  ||   : DataL DataL -> DataL [assoc ctor id: nilDL] .       concatenation
```

`nilDL` is the identity element of `||`, thanks to the Maude attribute `id:`. `assoc` states that `||` is associative. `ctor` states that `nilDL` and `||` are constructors of the sort `DataL`.

Several operators are introduced to model cryptographic primitives. For instance, the operators modeling the hash function, the sign and verify functions of ECC-based digital signature schemes are as follows:

```
op h        : DataL               -> Data .          hash function
op ecSign   : EsPriKey Data       -> Data .          ECC based signature: sign
                       sign digest
op ecVerify : EsPubKey Data Data  -> Bool .          ECC based signature: verify
```

where `EsPriKey` and `EsPubKey` are two sorts of the signing private keys and the verifying public keys. Given a digest, a verifying public key, and a signature, the following two equations define that `ecVerify` returns valid only if the signature is signed with the associated private key of the given public key (with the attached attribute `owise`, the second equation will not be applied unless the first equation cannot be applied):

```
eq ecVerify(pkes(SKES), ecSign(SKES,D), D) = true .
eq ecVerify(PKES,       SIGN,           D) = false [owise] .
```

5.3. Modeling the protocol execution

We suppose that there exists two honest users together with a dishonest user (the intruder) participating in the protocol. PQ OpenPGP is modeled as a state machine, where each state is in the form of a braced associative-commutative collection (AC-collection) of name-value pairs, which are called observable components in Maude.

We use the following observable components:

- (`ecdh`$[a]$: `< pk`$(sked)$ `;` $sked$ `>`): User a has an ECDH public/private key pair `pk`$(sked)$ and $sked$, where `< • ; • >` is defined as a pair of keys. Recall that the operator `pk` takes a private key as input and returns its associated public key as output. In fact, a user may have more than one ECDH public/private key pair, but for the sake of simplicity, in this analysis, we restrict each user to only one ECDH public/private key pair.

- (`ecsig`$[a]$: `< pkes`$(skes)$ `;` $skes$ `>`): User a has an ECC-based signature scheme's public/private key pair `pkes`$(skes)$ and $skes$, where `pk`$(skes)$ denotes the associated public key of $skes$.

- (`kyber`$[a]$: `< keygen`$(skkb)$ `;` $skkb$ `>`): User a has a Kyber public/private key pair `keygen`$(skkb)$ and $skkb$.

- (`dilit`$[a]$: `< pkdi`$(skdi)$ `;` $skdi$ `>`): User a has a Dilithium signature public/private key pair `pkdi`$(skdi)$ and $skdi$, where `pkdi`$(skdi)$ is the associated public key of $skes$.

- (`rd-sesskey`: ks): ks is a space-separated list of available session keys to be used by any user. That is, whenever a user needs to randomly generate a session key to encrypt a raw message, the head value of ks will be used. Note that we define ks as a list instead of a set to reduce the number of states generated in the checking experiments later on.

- (`e-knl`: ds): The intruder's knowledge is ds, which is a set of `Data`'s terms separated by `;`. ds includes all information that is available to the intruder, such as those grasped from messages exchanged between other users, and those synthesized by using cryptographic primitives.

- (`nw`: ems): The network, i.e., the collection of messages exchanged, is ems. Each encrypted message is in the form of `msg`$(a, b, cipher)$, where a, b, and $cipher$ respectively are the sender, the recipient, and the message content.

- (`ms`: ms): ms is a space-separated list of raw messages to be sent by users.

- (`rd-ecdh`: $skeds$): $skeds$ is a list of available values to be used as ECDH ephemeral private keys. That is, whenever a user queries a new ECDH ephemeral private key, the head value of $skeds$ will be used.

- (`rd-kyber`: $skkbs$): $skkbs$ is a list of available values to be used as Kyber ephemeral private keys.

- (`used-kyber`$[a]$: $skkbs$): $skkbs$ is a set of Kyber ephemeral private keys that was used by user a and was erased from a's memory.

The initial state is defined as follows:

```
eq init =
 {(ecsig[a]: (< pkes(skes1) ; skes1 >))
  (dilit[a]: (< pkdi(skdi1) ; skdi1 >))
  (ecdh[a] : (< pk(sked1) ; sked1 >))     (ecdh[b] : (< pk(sked2) ; sked2 >))
  (kyber[b]: (< keygen(skkb1) ; skkb1 >)) (kyber[a]: (< keygen(skkb2) ; skkb2 >))
  (ecsig[eve]: (< pkes(skes3) ; skes3 >)) (dilit[eve]: (< pkdi(skdi3) ; skdi3 >))
  (ecdh[eve] : (< pk(sked3) ; sked3 >))   (kyber[eve]: (< keygen(skkb3) ; skkb3 >))
  (rd-sesskey: (k1 k2))  (rd-ecdh: sked4) (rd-kyber: skkb4)
  (e-knl: (pkes(skes1) ; pkdi(skdi1) ; pk(sked1) ; pk(sked2) ;
           keygen(skkb1) ; pkes(skes3) ; skes3 ; pkdi(skdi3) ; skdi3 ;
           pk(sked3) ; sked3 ; keygen(skkb3) ; skkb3))
  (nw: empty)  (ms: (m1 m2))
  (used-kyber[b]: empty) (used-kyber[a]: empty)} .
```

`a` and `b` are the two honest users, while `eve` is the dishonest one. We suppose that `b` always be the recipient, while `a` always be the sender, and so two key pairs of ECC-based and Dilithium signatures are given to `a`. Initially, the intruder's knowledge includes their own keys and all other participants' public keys.

Encryption and sending a message is modeled by the following rewrite rule:

```
crl [send] :
   {(ms: (M MS)) (rd-sesskey: (K KS))
    (ecsig[A]: (< PKES ; SKES > , SKS))  (dilit[A]: (< PKDI ; SKDI > , SKS2))
    (ecdh[B] : (< PKED ; SKED > , SKS3)) (ecdh[A] : (< PKED2 ; SKED2 > , SKS4))
    (kyber[B]: (< PKKB ; SKKB > , SKS5)) (kyber[A] : (< PKKB2 ; SKKB2 > , SKS6))
    (nw: NW) (e-knl: DS) (used-kyber[A]: SKS7) OCs}
=> {(ms: MS)     (rd-sesskey: KS)
    (ecsig[A]: (< PKES ; SKES > , SKS))  (dilit[A]: (< PKDI ; SKDI > , SKS2))
    (ecdh[B] : (< PKED ; SKED > , SKS3)) (ecdh[A] : (SKS4))
    (kyber[B]: (< PKKB ; SKKB > , SKS5)) (kyber[A] : (SKS6))
    (nw: (msg(A,B, PKED2 || KBC || KC || C2) NW))
    (e-knl: (DS ; PKED2 ; KBC ; KC ; C2))
    (used-kyber[A]: (SKS7 , < PKKB2 ; SKKB2 >)) OCs}
if H    := h(M) /\
   SIGN := ecSign(SKES,H)  /\  SIGN2 := diSign(SKDI,H) /\
   EDSS := ss(PKED,SKED2)  /\  KBSS := encapsK(PKKB,SKKB2) /\
   KBC  := encapsC(PKKB,SKKB2) /\
   KEK  := kcombine(EDSS,PKED2,KBSS,KBC) /\
   KC   := senc(KEK, K) /\
   C2   := senc(K, SIGN || SIGN2 || M) .
```

`M` is the raw message being sent. Sender `A` first hashes `M`, obtaining `H`, and then signs `H` with his own ECC-based signature's and Dilithium's private keys, obtaining `SIGN` and `SIGN2`, respectively. `PKED` and `PKKB` are the ECDH's and Kyber's public keys of recipient `B`, while `SKED2` and `SKKB2` are the ECDH's and Kyber's ephemeral private keys of `A`. From these keys, `A` computes the two shared secrets `EDSS` and `KBSS` as well as the Kyber ciphertext `KBC`. The key-encryption-key `KEK` is then computed by the function `kCombine` (KeyCombine in Figure 5). `A` selects session key `K`, encrypts it with the key-encryption-key, obtaining `KC`, and encrypts `SIGN || SIGN2 || M` with

K, obtaining C2. The encrypted message msg(A,B, PKED2 || KBC || KC || C2) is put into the network in the successor state, namely, A sent that message to B. At the same time, the intruder learns the message content (the message content is put into e-knl), A erases his ECDH's and Kyber's ephemeral key pairs just used, and the Kyber's ephemeral private key is put into A's set of Kyber's ephemeral private keys used.

We suppose that the sender always deletes ECDH's and Kyber's ephemeral keys after sending a message. In contrast, the recipient is not forced to do so. Typically, users are required to periodically update the ephemeral keys. We defined two other rewrite rules modeling the process by which users generate new ephemeral key pairs.

5.4. Modeling the intruder

The intruder is given a capability of intercepting any message sent in the network to learn information carried in that message. This capability has been illustrated through the rewrite rule [send] previously presented. In addition, the intruder can (1) generate random session keys and raw messages to be sent, and (2) apply any cryptographic primitive function to derive new information from the information is available to them. Several rewrite rules are defined to model (2), some among them are as follows:

```
    hash anything
rl [hash] :
  {(e-knl: (M ; DS)) OCs} => {(e-knl: (M ; DS ; h(M))) OCs} .
    sign anything, SKES is any ECC based signature scheme's private key
rl [sign] :
   {(e-knl: (SKES ; h(D) ; DS)) OCs}
=> {(e-knl: (SKES ; h(D) ; DS ; ecSign(SKES,h(D)))) OCs} .
    compute keys and encapsulation by Encaps
    PKKB and SKKB are variables of Kyber public and private keys
crl [encaps] :
   {(e-knl: (PKKB ; SKKB ; DS)) OCs}
=> {(e-knl: (PKKB ; SKKB ; DS ; encapsC(PKKB, SKKB) ; encapsK(PKKB, SKKB))) OCs}
if PKKB =/= keygen(SKKB) .
```

With the rewrite rule [hash], the intruder is restricted to hash only raw messages instead of any terms of sort Data. Even though this restricts the intruder capabilities to some extent, it helps to reduce the number of states generated in the analysis later on. Recall that because we define ds in (e-knl: ds) as a set, there is no duplication in ds. As a result, it cannot be the case that an infinite number of different states are produced by applying the rule [hash] infinite times with the same message M.

We assume practical quantum computers are available to the intruder, from which they can break the security of ECDH and ECC-based signature schemes. If a public key of those two schemes is given to Eve, Eve can derive its associated private key. This capability is specified by the following two rewrite rules:

```
    breaking ECDH, Eve can derive private keys from public keys
rl [break-ecdh] :
   {(e-knl: (pk(SKED) ; DS)) OCs}   => {(e-knl: (pk(SKED) ; DS ; SKED)) OCs} .
    breaking ECC based signature schemes
```

```
rl [break-ecc-sign] :
   {(e-knl: (pkes(SKES) ; DS)) OCs} => {(e-knl: (pkes(SKES) ; DS ; SKES)) OCs} .
```

6. Formal analysis

We first check that the formal specification allows two users to successfully send an encrypted message and decrypt it. To this end, we define a Maude **search** command finding a state reachable from `init` such that an encrypted message sent from `A` to `B` exists in the network and `B` successfully decrypts and validates the message. Executing that **search** command, Maude almost immediately found such a state at the first depth, that is, only one rewrite rule (`send`) is applied, changing the state from `init` to the state found.

6.1. Secrecy of messages

All messages should be securely sent, that is, no one other than the sender and the recipient can learn the message content. This property can be called the *secrecy of messages*. To check this property, we define the following **search** command:

```
search [1,10] in PQOPENPGP : init =>*
 {(ecsig[A]: (< PKES ; SKES > , SKS))  (dilit[A]: (< PKDI ; SKDI > , SKS2))
  (ecdh[B] : (< PKED ; SKED > , SKS3)) (kyber[B]: (< PKKB ; SKKB > , SKS5))
  (nw: (msg(A,B, PKED2 || KBC || KC || C2) NW))
  (e-knl: (M ; DS)) OCs}
such that
  (A =/= eve and B =/= eve)   /\
  EDSS := ss(PKED2, SKED)     /\  KBSS := decaps(KBC, SKKB) /\
  KEK  := kcombine(EDSS, PKED2, KBSS, KBC) /\
  K    := sdec(KEK, KC) /\
  SIGN || SIGN2 || M := sdec(K, C2) /\
  ecVerify(PKES, SIGN,  h(M)) /\  diVerify(PKDI, SIGN2, h(M)) .
```

The command tries to find a state with bounded depth 10 in which there exists an encrypted message sent from `A` to `B` in the network, `B` successfully decrypts the raw message `M` and verifies the composite signatures, and `M` exists in Eve's knowledge. The procedure for the recipient `B` to decrypt and validate the message is as follows. `B` first computes (1) the ECDH shared secret `EDSS` from the received public key and her own private key; and (2) the Kyber shared secret `KBSS` by the function `Decaps` with the received encapsulation and her own private key. `B` then computes the key-encryption-key `KEK` and uses it to decrypt the session key `K`. `B` uses the session key to decrypt the encrypted message, obtaining a raw message `M` and two signatures. Finally, `B` verifies the two signatures.

Maude did not find such a state after about 1 minute and 39 seconds, meaning that the protocol enjoys the property up to depth 10. The experiment was conducted on a computing server that carries 192 GB of memory and a 2.8 GHz microprocessor with 8 cores. The experimental results are reported in Table 1 and will be discussed in detail below.

Property	Depth	Result	Time (h:m:s)	No. States
Secrecy of messages	8	∅	0:00:6.7	46317
	9	∅	0:00:22.2	98943
	10	∅	0:01:39	206972
	11	∅	0:08:31	430750
	12	∅	0:42:34	903344
	13	∅	5:08:16	1929731
Authenticity of messages	8	∅	0:06:40	46317
	9	∅	0:23:39	98943
	10	∅	1:26:30	206972
	11	∅	6:54:14	430750

Table 1
Experimental results. ∅ means that Maude did not find solution(s) for the given search command.

6.2. Learning ECDH shared secrets

We define another command to show that Eve can learn the ECDH shared secret after a message is exchanged between two users. This **search** command is identical to the previous command used to check the secrecy property of messages sent except that the observable component `e-knl` is now changed to the following:

```
(e-knl: (EDSS ; DS))
```

Recall that `EDSS` is the ECDH shared secret. Maude found a state after about 6 milliseconds. The found state is at depth 3 after the three following rewrite rules are applied:

1. rule `[send]`: `a` encrypts and sends a message to `b`.
2. rule `[break-ecdh]`: `eve` derives the ECDH ephemeral private key of `a` from the public key.
3. rule `[ss]`: `eve` computes the ECDH shared secret from `a`'s private key just obtained and `b`'s public key.

6.3. Authenticity of messages

This property states that if Bob successfully decrypts an encrypted message apparently sent from Alice and successfully verifies the composite signatures with Alice's verifying keys, obtaining a raw message M, then Alice indeed sent M to Bob. To check the property, we define the following **search** command:

```
search [1,10] in PQOPENPGP : init =>*
 {(ecsig[A]: (< PKES ; SKES > , SKS))   (dilit[A]: (< PKDI ; SKDI > , SKS2))
  (ecdh[B] : (< PKED ; SKED > , SKS3))  (kyber[B]: (< PKKB ; SKKB > , SKS5))
  (e-knl: (PKED2 ; KBC ; KC ; C2 ; DS)) (nw: NW)
  (used-kyber[A]: (< PKKB2 ; SKKB2 > , SKS6)) OCs}
such that
  (A =/= eve and B =/= eve)   /\
```

```
EDSS := ss(PKED2, SKED)     /\  KBSS := decaps(KBC, SKKB) /\
KEK  := kcombine(EDSS, PKED2, KBSS, KBC) /\
K    := sdec(KEK, KC) /\
SIGN || SIGN2 || M := sdec(K, C2) /\
ecVerify(PKES, SIGN,  h(M)) /\  diVerify(PKDI, SIGN2, h(M)) /\
not(msg(A,B, PKED2 || KBC || KC || C2) \in NW) /\
KBC == encapsC(PKKB, SKKB2) .
```

The command tries to find a state with bounded depth 10 such that:

(1) Eve knows four pieces of data `PKED2`, `KBC`, `KC`, and `C2` (Eve can use these pieces of data forming a message, and pretend `A` to send the message to `B`);

(2) from those four pieces of data, `B` successfully decrypts a raw message `M` and verifies the signatures with `A`'s verifying keys;

(3) `A` did not send those four pieces of data to `B`, namely, such a message does not exist in the network.

If such a state exists, it is possible to happen a case in which `B` receives a valid message apparently sent from `A`, but the message is actually forged by Eve. Maude did not find such a state after about 1 hour and 26 minutes. The **`search`** command bounds `KBC` to be the ciphertext encapsulating an ephemeral private key of `A`, meaning that the ciphertext was really created by `A`. As mentioned before, we suppose that the binding between a public key (or KEM ciphertext) and its real owner is always correctly done.

Table 1 shows the experimental results of (1) the *secrecy of messages* from depth 8 to depth 13 and (2) the *authenticity of messages* from depth 8 to depth 11. Those results confirm that PQ OpenPGP enjoys (1) and (2) up to depths 13 and 11, respectively. With the same depth, checking (1) is significantly faster than checking (2), which is mostly because, in each state, there is a huge number of substitutions for the following pattern in the **`search`** command of (2):

```
(e-knl: (PKED2 ; KBC ; KC ; C2 ; DS))
```

`KC` and `C2` are variables of the sort `Data`, and so they can be substituted by any terms of `Data` or its subsorts. Besides, `;` is AC, making the number of substitution solutions increase.

7. Concluding remarks

Using the Maude tool, we have presented a formal analysis of the OpenPGP's post-quantum extension. The protocol is modeled as a state machine with three participants, one of whom is an active attacker. The experimental results have confirmed that the protocol enjoys two properties: *secrecy of messages* and *authenticity of messages* up to some specific depths. Even though we limit the number of protocol participants to three, the number of the state space generated is really huge (nearly 1 million states at depth 12), which makes us unable to proceed with the experiments at deeper depths due to time limitations. State explosion, however, is not dedicated to Maude only, but rather the burdensome issue of all analysis/verification approaches based on model checking.

Interactive theorem proving, the complementary approach to model checking, is one possible way to escape from the state explosion issue. Protocol analysis/verification based on interactive theorem proving makes it feasible to deal with an infinite state space. In particular, with cryptographic protocols, we can do formal verification with an unbounded number of participants and session executions. However, this comes at a cost: human creativity is required. Verification by using the tool IPSG [29] to produce the so-called proof scores [30] is a promising way that we plan to apply the method to the OpenPGP's post-quantum extension.

References

[1] P. Wouters, D. Huigens, J. Winter, N. Yutaka, OpenPGP, Internet-Draft draft-ietf-openpgp-crypto-refresh-10, Internet Engineering Task Force, 2023. URL: https://datatracker.ietf.org/doc/draft-ietf-openpgp-crypto-refresh/10/, work in Progress.

[2] H. Finney, L. Donnerhacke, J. Callas, R. L. Thayer, D. Shaw, OpenPGP Message Format, RFC 4880, 2007. doi:10.17487/RFC4880.

[3] A. Jivsov, Elliptic Curve Cryptography (ECC) in OpenPGP, RFC 6637, 2012. doi:10.17487/RFC6637.

[4] P. Shor, Algorithms for quantum computation: discrete logarithms and factoring, in: Proceedings 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 124–134. doi:10.1109/SFCS.1994.365700.

[5] National Institute of Standards and Technology (NIST), Post-quantum cryptography, NIST, 2017. URL: https://csrc.nist.gov/projects/post-quantum-cryptography.

[6] S. Kousidis, F. Strenzke, A. Wussler, Post-Quantum Cryptography in OpenPGP, Internet-Draft draft-wussler-openpgp-pqc-02, Internet Engineering Task Force, 2023. URL: https://datatracker.ietf.org/doc/draft-wussler-openpgp-pqc/02/, work in Progress.

[7] J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, D. Stehlé, CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM, in: 2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018, IEEE, 2018, pp. 353–367. doi:10.1109/EuroSP.2018.00032.

[8] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, D. Stehlé, CRYSTALS-Kyber: Algorithm Specifications And Supporting Documentation (version 3.02), 2021. URL: https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf.

[9] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, D. Stehlé, CRYSTALS-Dilithium – Algorithm Specifications and Supporting Documentation (Version 3.1), 2021. URL: https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf.

[10] F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer, R. Rubio, C. L. Talcott, Programming and symbolic computation in maude, J. Log. Algebraic Methods Program. 110 (2020). doi:10.1016/j.jlamp.2019.100497.

[11] D. Dolev, A. C. Yao, On the security of public key protocols, IEEE Trans. Inf. Theory 29 (1983) 198–207.

[12] S. Gazdag, S. Grundner-Culemann, T. Guggemos, T. Heider, D. Loebenberger, A formal analysis of IKEv2's post-quantum extension, in: ACSAC '21: Annual Computer Security

Applications Conference, Virtual Event, USA, December 6 - 10, 2021, ACM, 2021, pp. 91–105. doi:10.1145/3485832.3485885.

[13] S. Frankel, S. Krishnan, IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap, RFC 6071, 2011. doi:10.17487/RFC6071.

[14] V. Smyslov, Intermediate Exchange in the Internet Key Exchange Protocol Version 2 (IKEv2), RFC 9242, 2022. doi:10.17487/RFC9242.

[15] C. Tjhai, M. Tomlinson, G. Bartlett, S. Fluhrer, D. V. Geest, O. Garcia-Morchon, V. Smyslov, Multiple Key Exchanges in the Internet Key Exchange Protocol Version 2 (IKEv2), RFC 9370, 2023. doi:10.17487/RFC9370.

[16] D. A. Basin, C. Cremers, J. Dreier, R. Sasse, Symbolically analyzing security protocols using TAMARIN, ACM SIGLOG News 4 (2017) 19–30. doi:10.1145/3157831.3157835.

[17] A. Hülsing, K. Ning, P. Schwabe, F. Weber, P. R. Zimmermann, Post-quantum wireguard, in: 2021 IEEE Symposium on Security and Privacy, 2021, pp. 304–321. doi:10.1109/SP40001.2021.00030.

[18] J. A. Donenfeld, Wireguard: Next generation kernel network tunnel, in: 24th Annual Network and Distributed System Security Symposium, NDSS 2017, 2017. https://www.wireguard.com/papers/wireguard.pdf.

[19] D. D. Tran, K. Ogata, S. Escobar, S. Akleylek, A. Otmani, Kyber, Saber, and SK-MLWR Lattice-Based Key Encapsulation Mechanisms Model Checking with Maude, IET Information Security 2023 (2023) 9399887. doi:10.1049/2023/9399887.

[20] V. García, S. Escobar, K. Ogata, S. Akleylek, A. Otmani, Modelling and verification of post-quantum key encapsulation mechanisms using maude, PeerJ Comput. Sci. 9 (2023) e1547. doi:10.7717/PEERJ-CS.1547.

[21] E. Rescorla, T. Dierks, The Transport Layer Security (TLS) Protocol Version 1.2, RFC 5246, 2008. doi:10.17487/RFC5246.

[22] M. Campagna, E. Crockett, Hybrid Post-Quantum Key Encapsulation Methods (PQ KEM) for Transport Layer Security 1.2 (TLS), RFC, RFC Editor, 2021. URL: https://datatracker.ietf.org/doc/html/draft-campagna-tls-bike-sike-hybrid.

[23] D. D. Tran, C. M. Do, S. Escobar, K. Ogata, Hybrid Post-Quantum Transport Layer Security formal analysis in Maude-NPA and its parallel version, PeerJ Comput. Sci. (2023). To appear.

[24] S. Escobar, C. Meadows, J. Meseguer, A Rewriting-Based Inference System for the NRL Protocol Analyzer and Its Meta-Logical Properties, Theor. Comput. Sci. 367 (2006) 162–202. doi:10.1016/j.tcs.2006.08.035.

[25] C. M. Do, A. Riesco, S. Escobar, K. Ogata, Parallel Maude-NPA for cryptographic protocol analysis, in: K. Bae (Ed.), Rewriting Logic and Its Applications - 14th International Workshop, WRLA@ETAPS 2022, Munich, Germany, April 2-3, 2022, Revised Selected Papers, volume 13252 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 253–273. doi:10.1007/978-3-031-12441-9 13.

[26] S. Escobar, J. Meseguer, P. Thati, Narrowing and rewriting logic: from foundations to applications, in: F. J. López-Fraguas (Ed.), Proceedings of the 15th Workshop on Functional and (Constraint) Logic Programming, WFLP 2006, Madrid, Spain, November 16-17, 2006, volume 177 of *Electronic Notes in Theoretical Computer Science*, Elsevier, 2006, pp. 5–33. doi:10.1016/j.entcs.2007.01.004.

[27] F. J. Thayer, J. C. Herzog, J. D. Guttman, Strand spaces: Why is a security protocol correct?, in: Security and Privacy - 1998 IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 3-6, 1998, Proceedings, IEEE Computer Society, 1998, pp. 160–171. doi:10.1109/SECPRI.1998.674832.

[28] J.-P. Aumasson, D. J. Bernstein, W. Beullens, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, A. Hülsing, P. Kampanakis, S. Kölbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, P. Schwabe, B. Westerbaan, SPHINCS+ – Submission to the 3rd round of the NIST post-quantum project. v3.1, 2022. URL: https://sphincs.org/data/sphincs+-r3.1-specification.pdf.

[29] D. D. Tran, K. Ogata, Formal verification of TLS 1.2 by automatically generating proof scores, Computers & Security 123 (2022) 102909. doi:https://doi.org/10.1016/j.cose.2022.102909.

[30] K. Ogata, K. Futatsugi, Proof scores in the OTS/CafeOBJ method, in: Formal Methods for Open Object-Based Distributed Systems, 6th IFIP WG 6.1 International Conference, FMOODS 2003, Paris, France, volume 2884 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 170–184. doi:10.1007/978-3-540-39958-2\ 12.

Automated Quantum Program Verification in Probabilistic Dynamic Quantum Logic

Canh Minh Do[1,*], Tsubasa Takagi[2] and Kazuhiro Ogata[1]

[1]Japan Advanced Institute of Science and Technology, 1-8 Asahidai, Nomi, Japan
[2]Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro-ku, Tokyo, Japan

Abstract

We proposed an automated approach to verifying quantum programs based on Basic Dynamic Quantum Logic (BDQL), a simplified version of Dynamic Quantum Logic, and developed a support tool in Maude, a specification/programming language based on rewriting logic, to automate the verification process. This paper extends BDQL to Probabilistic Dynamic Quantum Logic (PDQL) to verify probabilistic quantum programs by introducing a probabilistic operator in the formulas of PDQL. We also extend the support tool for BDQL to make a new support tool for PDQL to automatically verify both qualitative and quantitative properties for quantum programs with PDQL. As case studies, we use our support tool to verify several quantum programs: Superdense Coding, Quantum Teleportation, Quantum Secret Sharing, Entanglement Swapping, Quantum Gate Teleportation, Quantum Relay Scheme, Bidirectional Quantum Teleportation, Two-qubit Quantum Teleportation, and Quantum Network Coding, where the probability is intentionally expressed in the formalizations of quantum programs, demonstrating the expressiveness of PDQL.

Keywords

Probabilistic Dynamic Quantum Logic, Quantum Programs, Quantum Protocols, Dirac notation, Maude

1. Introduction

Quantum computing holds the potential to revolutionize various fields by harnessing the principles of quantum mechanics to perform computations that are currently infeasible for classical computers. For example, in the field of cryptography, the arrival of future quantum computers has the potential to break widely used public-key cryptography methods, such as Rivest–Shamir–Adleman (RSA) and Elliptic-curve Cryptography (ECC), by effectively factoring large numbers or solving the discrete logarithm problem using Shor's fast algorithm [1]. Quantum computing uses different principles of quantum mechanics, such as superposition, entanglement, and measurement, which are completely different from classical computing. As a result, it is challenging to accurately design and implement quantum algorithms, programs, and protocols. Therefore, it is crucial to ensure the correctness of quantum systems through verification.

The 2nd International Workshop on Formal Analysis and Verification of Post-Quantum Cryptographic Protocols, November 11, 2023, Brisbane, Australia

*Corresponding author.
canhdo@jaist.ac.jp (C. M. Do); takagi.t.ah@m.titech.ac.jp (T. Takagi); ogata@jaist.ac.jp (K. Ogata)
0000-0002-1601-4584 (C. M. Do); 0000-0001-9890-1015 (T. Takagi); 0000-0002-4441-3259 (K. Ogata)

Dynamic Logic [2] has been playing a significant role in the formal verification of classical systems for some decades. To handle quantum effects, Dynamic Quantum Logic [3], the quantum counterpart of Dynamic Logic, has been developed. However, we lack an automated approach to verifying quantum programs using Dynamic Quantum Logic. Our research group devised an automated approach to verifying quantum programs using Basic Dynamic Quantum Logic (BDQL) [4], a simplified version of Dynamic Quantum Logic, and developed a support tool in Maude [5], a specification/programming language based on rewriting logic [5], to automate the verification process by incorporating symbolic reasoning for quantum computation in [6]. This paper extends BDQL to Probabilistic Dynamic Quantum Logic (PDQL) to verify probabilistic quantum programs by introducing a probabilistic operator in the formulas of PDQL. We also extend the support tool for BDQL to make a new support tool for PDQL to automatically verify both qualitative and quantitative properties for quantum programs with PDQL. As case studies, we use our support tool to verify several quantum programs: Superdense Coding [7], Quantum Teleportation [8], Quantum Secret Sharing [9], Entanglement Swapping [10], Quantum Relay Scheme [11], Bidirectional Quantum Teleportation [12], Quantum Gate Teleportation [13], Two-qubit Quantum Teleportation [14], and Quantum Network Coding [15], where the probability is intentionally expressed in the formalizations of quantum programs, demonstrates the expressiveness of PDQL. The support tool and case studies are publicly available at https://github.com/canhminhdo/DQL.

The rest of the paper is organized as follows: Section 2 describes Basic Dynamic Quantum Logic (BDQL); Section 3 presents Probabilistic Dynamic Quantum Logic (PDQL); Section 5 demonstrates the application of PDQL for probabilistic quantum program verification; Section 5 describes the implementation of PDQL; Section 6 exhibits the experimental results with our support tool for several quantum programs; Section 7 presents some existing work; and Section 8 concludes the paper with some pieces of future work.

2. Basic Dynamic Quantum Logic

This section formulates Basic Dynamic Quantum Logic (BDQL) [4]. 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, non-deterministic choice, and test, respectively.

The syntax of BDQL is exactly the same as that of Propositional Dynamic Logic (PDL) [2] 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 [16]. The set $\mathcal{C}(\mathcal{H})$ is called a Hilbert lattice [17] 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 2.1. 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 2.1. $[\![\ ]\!]^M$ is well-defined. That is, $[\![A]\!]^M \in \mathcal{C}(\mathcal{H})$ for each $A \in L$.

Proof. See our proof in [4]. □

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 nondeterministic choice operator, and ? is the quantum test operator and is used to represent a result of projective measurement (see Section 4.1).

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 2.2. 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 [18] 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 2.1 does not hold).

The following theorem gives the theoretical background for rewriting the statement of the form $(M, s) \models A$ in our implementation explained in Section 4 of our previous work [4].

Theorem 2.2. 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. Probabilistic Dynamic Quantum Logic

In general, the outcome of a projective measurement is not determined in advance because there is a probability associated with each outcome. To capture this probabilistic ingredient, we employ a probabilistic operator $\mathbf{P}^{\geq r}$ from [3] to formulate Probabilistic Dynamic Quantum Logic (PDQL) as follows:

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

where r denotes a rational number in the closed interval $[0, 1]$. The expression $\mathbf{P}^{\geq r}A$ means that a projective measurement of A on the current state of a quantum system will succeed with probability $\geq r$.

We can also use the probabilistic formula inside the dynamic modality for quantum tests as follows:

$$[A?^{\geq r}]B \triangleq \mathbf{P}^{\geq r}A \wedge [A?]B,$$

This means that if the quantum test $A?$ (or the projective measurement of A) succeeds with probability $\geq r$, then B will be the case after the successful execution of the quantum test.

Similarly, we can define other probabilistic operators $\mathbf{P}^{>r}$, $\mathbf{P}^{\leq r}$, $\mathbf{P}^{<r}$, $\mathbf{P}^{=r}$, and $\mathbf{P}^{\neq r}$. Regarding the binary projective measurement of A, we have the following axioms:

$$\begin{array}{lll} \mathbf{P}^{\geq r}A \to \mathbf{P}^{\leq r}\neg A, & \mathbf{P}^{>r}A \to \mathbf{P}^{<r}\neg A, & \mathbf{P}^{\leq r}A \to \mathbf{P}^{\geq (1-r)}\neg A, \\ \mathbf{P}^{<r}A \to \mathbf{P}^{>r}\neg A, & \mathbf{P}^{=r}A \to \mathbf{P}^{=(1-r)}A, & \mathbf{P}^{\neq r}A \to \mathbf{P}^{\neq (1-r)}. \end{array}$$

Notice that these axioms only hold for the binary projective measurement of A.

The function $[\![\ ]\!]^M : L \to \mathcal{C}(\mathcal{H})$ is extended to handle the probabilistic operator $\mathbf{P}^{\geq r}$ using the Born rule as follows:

$$s \in [\![\mathbf{P}^{\geq r}A]\!]^M \text{ if and only if } \left\langle s \middle| P_{[\![A]\!]^M}(s) \right\rangle \geq r,$$

where $P_{[\![A]\!]^M}(s)$ is the projection of the current state s on the closed linear subspace spanned by $[\![A]\!]^M$, and $\langle s|s'\rangle$ is the inner product of two vectors representing two states s and s', respectively. Therefore, we write $(M, s) \models \mathbf{P}^{\geq r}A$ if and only if $s \in [\![\mathbf{P}^{\geq r}A]\!]^M$ as usual.

The following theorem gives the theoretical background for rewriting the statement involving the probabilistic operator for quantum tests.

Theorem 3.1. The following holds for any M, $s \in \mathcal{H}$, and $r \in [0, 1]$.

1. $(M, s) \models [A?^{\geq r}]B$, if and only if $(M, s) \models \mathbf{P}^{\geq r}A$ and $(M, s) \models [A?]B$.
2. $(M, s) \models [A?^{>r}]B$, if and only if $(M, s) \models \mathbf{P}^{>r}A$ and $(M, s) \models [A?]B$.
3. $(M, s) \models [A?^{\leq r}]B$, if and only if $(M, s) \models \mathbf{P}^{\leq r}A$ and $(M, s) \models [A?]B$.
4. $(M, s) \models [A?^{<r}]B$, if and only if $(M, s) \models \mathbf{P}^{<r}A$ and $(M, s) \models [A?]B$.
5. $(M, s) \models [A?^{=r}]B$, if and only if $(M, s) \models \mathbf{P}^{=r}A$ and $(M, s) \models [A?]B$.
6. $(M, s) \models [A?^{\neq r}]B$, if and only if $(M, s) \models \mathbf{P}^{\neq r}A$ and $(M, s) \models [A?]B$.

Proof. Straightforward. □

4. Application to Probabilistic Quantum Program Verification

This section describes the behavior and desired properties of some specific quantum programs in the language of PDQL. These properties can be verified automatically using our support tool as shown in Sections 5 and 6.

4.1. 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 $|c_0|^2 + |c_1|^2 = 1$. This notation of vectors is called bra-ket notation (also called Dirac notation). $|\psi\rangle$ is called a ket vector. The bra vector $\langle\psi|$ is defined as a row vector whose elements are complex conjugates of the elements of the corresponding ket vector $|\psi\rangle$. 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 $|11\rangle = |1\rangle \otimes |1\rangle$.

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 [19]. 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 $|c_1|^2$.

4.2. Standard Interpretation

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

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

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

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

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 functionato 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. $\{|\Psi\rangle\}$).

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 for quantum tests with probability in PDQL:

$$\textbf{if } \mathbf{P}^{\geq r} A \textbf{ then } a \textbf{ else } b \textbf{ fi} = (A?^{\geq r}\ ;\ a) \cup (\neg A?^{\leq (1-r)}\ ;\ b).$$

This program means the selection depends on the outcomes of projective measurement with respect to the probabilities. That is, execute a or b depending on the result of the measurement $\{P_{[\![A]\!]^M}, P_{[\![\neg A]\!]^M}\}$ with respect to $\mathbf{P}^{\geq r}A$ and $\mathbf{P}^{\leq (1-r)}\neg A$.

4.3. Case Studies

In the sequel, we use PDQL to verify some case studies involving probability, which intentionally is added to the formalizations of quantum programs to demonstrate the expressiveness of PDQL. For the sake of brevity, this section only describes some quantum programs, including Quantum Relay Scheme [11], Bidirectional Quantum Teleportation [12], Two-qubit Quantum Teleportation [14], and Quantum Network Coding [15]. Meanwhile, other quantum programs, including Superdense Coding [7], Quantum Teleportation [8], Quantum Secret Sharing [9], Entanglement Swapping [10], and Quantum Gate Teleportation [13], can be derived in a similar way. The reader interested in them is referred to our previous work [4] for their description without probability.

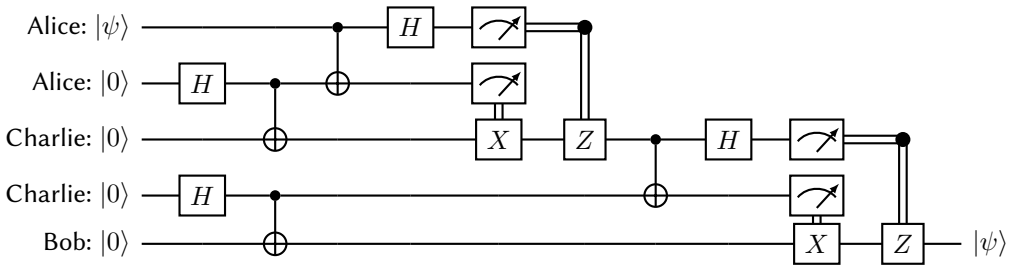


Figure 1: Quantum Relay Scheme

Quantum Relay Scheme

Quantum Relay Scheme [11] allows us to teleport a quantum state from one quantum device to another wirelessly even though these two devices do not share EPR pairs mutually, which differs from Quantum Teleportation [8]. The quantum circuit of Quantum Relay Scheme is depicted in Figure 1. The program of Quantum Relay Scheme with probability intentionally added is described as follows:

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

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

$$(\overline{M}_5, |\psi\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle) \models [\mathbf{relay}]p(4,|\psi\rangle).$$

Bidirectional Quantum Teleportation

Bidirectional Quantum Teleportation [12] is based on EPR pairs and entanglement swapping to allow two users to simultaneously transmit an unknown single qubit to each other. The quantum circuit of Bidirectional Quantum Teleportation is depicted in Figure 2. The program of Bidirectional Quantum Teleportation with probability intentionally added is described as follows:

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

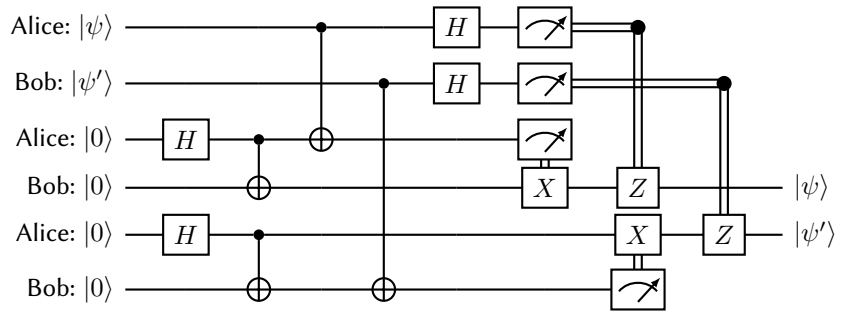


Figure 2: Bidirectional Quantum Teleportation

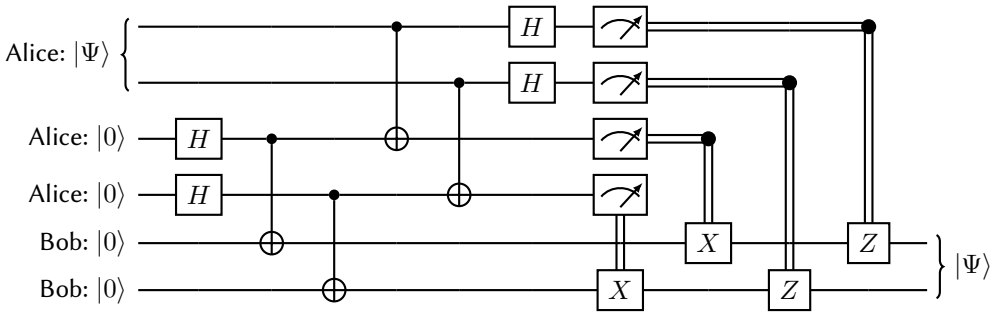


Figure 3: Two-qubit Quantum Teleportation

$$; \textbf{if } p(1, |0\rangle)^{\geq 1/2} \textbf{ then skip else } \mathtt{Z}(4) \textbf{ fi}.$$

The desired property of Bidirectional Quantum Teleportation is that "two pure states $|\psi\rangle$ and $|\psi'\rangle$ owned by two users are correctly teleported to each other." In PDQL, this property is expressed as follows:

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

Two-qubit Quantum Teleportation

Two-qubit Quantum Teleportation [14] allows us to teleport an arbitrary two-qubit pure states $|\Psi\rangle = c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle$ from one source to another, where $|c_{00}|^2 + |c_{01}|^2 + |c_{10}|^2 + |c_{11}|^2 = 1$. The quantum circuit of Two-qubit Quantum Teleportation is depicted in Figure 3. The program of Two-qubit Quantum Teleportation with probability intentionally added is described as follows:

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

$$; \textbf{if } p(0,|0\rangle)^{\geq 1/2} \textbf{ then skip else } \mathtt{Z}(4) \textbf{ fi}.$$

The desired property of Two-qubit Quantum Teleportation is that "arbitrary two-qubit pure states $|\Psi\rangle$ is correctly teleported." In PDQL, this property is expressed as follows:

$$(\overline{M}_6, |\Psi\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle) \models [\textbf{twoTeleport}]p(4,5,|\Psi\rangle).$$

Quantum Network Coding

Quantum Network Coding [15] enables the simultaneous generation of EPR pairs by using only local operations and classical communication (LOCC) and the sharing of EPR pairs between adjacent quantum repeaters. Simultaneous quantum communication can be achieved by using Teleportation with these EPR states. This facilitates the transmission of quantum information on complex networks at higher rates than straightforward routing strategies, such as the simple entanglement swapping strategy. For the sake of simplicity, we do not depict the quantum circuit of Quantum Network Coding here, while the reader is referred to their original paper [15]. The protocol introduces two new techniques called "Connection" and "Removal" to manipulate entangled states and systematize the methods of encoding, which are described as follows:

$$\begin{aligned}
\textbf{con}(N1,(N2 \to N3)) &= \mathtt{CX}(N1,N2) \\
&\quad ; \textbf{ if } p(N2,|0\rangle)^{\geq 1/2} \textbf{ then skip else } \mathtt{X}(N3) \textbf{ fi}. \\
\textbf{fanout}(N1,(N2 \to N3),(N4 \to N5)) &= \mathtt{con}(N1,(N2 \to N3)) \ ; \mathtt{con}(N1,(N4 \to N5)). \\
\textbf{add}(N1,N2,(N3 \to N4)) &= \mathtt{CX}(N1,N3) \ ; \mathtt{CX}(N2,N3) \\
&\quad ; \textbf{ if } p(N3,|0\rangle)^{\geq 1/2} \textbf{ then skip else } \mathtt{X}(N4) \textbf{ fi}. \\
\textbf{rem}(N1 \to N2) &= \mathtt{H}(N1) \\
&\quad ; \textbf{ if } p(N1,|0\rangle)^{\geq 1/2} \textbf{ then skip else } \mathtt{Z}(N2) \textbf{ fi}. \\
\textbf{remadd}(N3 \to (N1,N2)) &= \mathtt{H}(N3) \\
&\quad ; \textbf{ if } p(N3,|0\rangle)^{\geq 1/2} \textbf{ then skip} \\
&\quad \textbf{else } \mathtt{Z}(N1) \ ; \mathtt{Z}(N2) \textbf{ fi} \ .
\end{aligned}$$

where $N1$, $N2$, $N3$, and $N4$ are natural numbers in the range $[0,9]$. Then, the program of Quantum Network Coding with probability intentionally added is described via **con**, **fanout**, **add**, **rem**, and **remadd** as follows:

$$\begin{aligned}
\textbf{networkcoding} = \ & \mathtt{H}(0) \ ; \mathtt{CX}(0,1) \ ; \mathtt{H}(2) \ ; \mathtt{CX}(2,3) \\
& ; \mathtt{H}(4) \ ; \mathtt{CX}(4,5) \ ; \mathtt{H}(6) \ ; \mathtt{CX}(6,7) \\
& ; \mathtt{H}(8) \ ; \mathtt{CX}(8,9) \ ; \mathtt{H}(10) \ ; \mathtt{CX}(10,11) \\
& ; \mathtt{H}(12) \ ; \mathtt{CX}(12,13) \\
& ; \mathtt{con}(0,(2 \to 3) \ ; \mathtt{con}(4,(6 \to 7)) \ ; \mathtt{add}(3,7,(8 \to 9)) \\
& ; \mathtt{fanout}(9,(10 \to 11),(12 \to 13)) \ ; \mathtt{CX}(13,5) \ ; \mathtt{CX}(11,1) \\
& ; \mathtt{rem}(11 \to 9) \ ; \mathtt{rem}(13 \to 9) \ ; \mathtt{remadd}(9 \to (3,7))
\end{aligned}$$

$$; \texttt{rem}(3 \to 0) \; ; \texttt{rem}(7 \to 4)$$
$$; \texttt{SWAP}(2, 4) \; ; \texttt{SWAP}(0, 4).$$

The desired property of Quantum Network Coding is that "the two EPR states (in this case $|\mathrm{EPR}\rangle$) are simultaneously generated." In PDQL, this property is expressed as follows:

$$(\overline{M}_{14}, |0\rangle^{\otimes 14}) \models [\mathbf{networkcoding}]p(1, 2, |\mathrm{EPR}\rangle) \wedge p(4, 5, |\mathrm{EPR}\rangle).$$

5. Implementation of Probabilistic Dynamic Quantum Logic

This section describes the implementation of PDQL in Maude [5], a specification/programming language based on rewriting logic [20]. Hence, the notations used in this section follow the Maude syntax.

5.1. Syntax of Probabilistic Dynamic Quantum Logic

Recalling *the syntax of Basic Dynamic Quantum Logic* (BDQL) in our previous work [4], we defined two sorts `AtomicProg` and `Prog` for atomic programs Π_0 and star-free regular programs Π, respectively, where `AtomicProg` is a subsort of `Prog`. Additionally, we defined several operators for atomic programs: `I(_)`, `H(_)`, `X(_)`, `Y(_)`, and `Z(_)`. These 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, and Z, will be applied, respectively. Furthermore, `CX(_,_)` and `SWAP(_,_)` operators take two natural numbers as inputs, denoting the indices of two qubits of a pure state on which CX and $SWAP$ gates will be applied. We also defined `abort`, `skip`, `_;_` , `_U_`, and `_?` operators to construct star-free regular programs in BDQL, as specified in Section 2.

We defined two sorts `AtomicFormula` and `Formula` for atomic formulas L_0 and general formulas L in BDQL, respectively, where `AtomicFormula` is a subsort of `Formula`. Additionally, we defined several operators for constructing formulas in BDQL: `P(_,_)` and `P(_,_,_)` operators are atomic formulas representing projections of the forms $p(i, |\psi\rangle)$ and $p(i, j, |\Psi\rangle)$, respectively; and the `neg_`, `_/\_`, and `[_]_` operators to construct formulas in BDQL, as specified in Section 2. Moreover, the `if_then_else_fi` command in Section 4.2 was also implemented for convenient use.

Regarding *the syntax of Probabilistic Dynamic Quantum Logic* (PDQL), we extend the syntax of BDQL by introducing the probabilistic operator as follows:

```
op _with prob >=_  : Formula Scalar -> Formula [ctor] .
op _with prob >_   : Formula Scalar -> Formula [ctor] .
op _with prob <=_  : Formula Scalar -> Formula [ctor] .
op _with prob <_   : Formula Scalar -> Formula [ctor] .
op _with prob ==_  : Formula Scalar -> Formula [ctor] .
op _with prob =/=_ : Formula Scalar -> Formula [ctor] .
```

where the five operators above explicitly denote the probabilistic operators $\mathbf{P}^{\geq r}A$, $\mathbf{P}^{>r}A$, $\mathbf{P}^{\leq r}A$, $\mathbf{P}^{<r}A$, $\mathbf{P}^{=r}A$, $\mathbf{P}^{\neq r}A$, respectively, and r is a rational number $\in [0, 1]$. However, we define r as belonging to a sort of Scalar in our formalization because we also want to reason on symbolic complex constants included in the value of probability.

5.2. Semantics of Probabilistic Dynamic Quantum Logic

We extend the semantics of BDQL to describe the semantics of PDQL by introducing the following equations, which strictly follow Theorem 3.1. For the sake of simplicity, we only describe equations used for `_with prob >=_` operator representing the probabilistic operator $\mathbf{P}^{\geq r} A$, while other operators are similar.

We define some Maude variables before use as follows: `Q` is a Maude variable of quantum states, `Phi` is a Maude variable of formulas, `Prob` is a Maude variable of scalars, `M` is a Maude variable of matrices, and `N`, `N1`, and `N2` is Maude variables of natural numbers.

We first handle the probabilistic operator in a formula as follows:

```
ceq Q |= P(N, M) with prob >= Prob = emptyJS if (Q).Prob(N, M) .>= Prob .
ceq Q |= P(N1, N2, M) with prob >= Prob = emptyJS if (Q).Prob(N1, N2, M) .>= Prob.
```

where `emptyJS` denote success in checking and `.>=` operator denotes "greater than or equal to" when comparing two complex numbers.

We then handle the probabilistic operator in the dynamic modality as follows:

```
ceq Q |= [(P(N, M) with prob >= Prob)?] Phi = Q |= [P(N, M)?] Phi
                                          if (Q).Prob(N, M) .>= Prob .
```

With respect to the equations specified in our support tool, $(M, s) \models A$ if `s |= A` is simplified to `emptyJS`. Otherwise, $(M, s) \not\models A$ if `s |= A` is not simplified to `emptyJS`.

6. Experimental Results

In this section, we illustrate the application of our support tool for verifying Quantum Relay Scheme in Maude, using it as an example. Similar procedures can be done for verifying other quantum programs, which are publicly available at https://github.com/canhminhdo/DQL. Following that, we summarize the experimental results for several quantum programs employed in our experiments, including Superdense Coding [7], Quantum Teleportation [8], Quantum Secret Sharing [9], Entanglement Swapping [10], Quantum Relay Scheme [11], Bidirectional Quantum Teleportation [12], Two-qubit Quantum Teleportation [14], Quantum Gate Teleportation [13], and Quantum Network Coding [15].

Assuming that `RELAY-SCHEME` represents the specification of Quantum Relay Scheme, `initQState` represents for the initial state for `RELAY-SCHEME`, and `qubitAt` denotes the function to retrieve a single qubit at a specific index. We can verify the correctness of Quantum Relay Scheme using our support tool by executing the following `reduce` command in Maude:

```
red in RELAY-SCHEME : initQState |= [
    H(1) ; CX(1, 2) ; H(3) ; CX(3, 4) ; CX(0, 1) ; H(0) ;
    if P(1, |0>) with prob >= 1/2 then skip else X(2) fi ;
    if P(0, |0>) with prob >= 1/2  then skip else Z(2) fi ;
    CX(2, 3) ; H(2) ;
    if P(3, |0>) with prob >= 1/2  then skip else X(4) fi ;
    if P(2, |0>) with prob >= 1/2  then skip else Z(4) fi
] P(4, qubitAt(initQState, 0)) .
```

Table 1
Experimental results with our support tool for several quantum programs in PDQL

Protocol	Qubits	Rewrite Steps	Verification Time
Superdense Coding	2	2,451	1ms
Quantum Teleportation	3	9,034	4ms
Quantum Secret Sharing	4	39,041	18ms
Entanglement Swapping	4	14,272	6ms
Quantum Relay Scheme	5	44,939	26ms
Bidirectional Quantum Teleportation	6	47,717	27ms
Two-qubit Quantum Teleportation	6	660,313	238ms
Quantum Gate Teleportation	6	667,806	250ms
Quantum Network Coding	14	11,568,281	4,811ms

The `reduce` command automatically performs an equational simplification process based on the equations defined in our tool. Within just a few moments, the command returns an `emptyJS` result. Consequently, this confirms the correctness of Quantum Relay Scheme with probability intentionally added by using our support tool, the implementation of PDQL in Maude. The property being verified informally says that the first qubit of the initial quantum state at index 0 is correctly teleported to the fifth qubit at index 4 in the final quantum state.

We conducted experiments on an iMac equipped with a 4 GHz microprocessor with eight cores and 32 GB memory of RAM. Table 1 presents the experimental results. We have successfully confirmed the correctness of Superdense Coding [7], Quantum Teleportation [8], Quantum Secret Sharing [9], Entanglement Swapping [10], Quantum Relay Scheme [11], Bidirectional Quantum Teleportation [12], Quantum Gate Teleportation [13], Two-qubit Quantum Teleportation [14], and Quantum Network Coding [15] in accordance with the specified properties outlined in Section 4. Comparing to our previous work [4], we extend our case studies by adding additional quantum programs to be verified with and without the probability. For all case studies from two to 14 qubits, we can quickly verify their correctness within just a few moments using our support tool despite the considerable number of rewrite steps involved. These results would have been extremely challenging to obtain without the assistance of computer programs like our support tool, especially in the case of Quantum Network Coding. This shows the usefulness of our automated approach for verifying quantum programs in PDQL, employing the symbolic approach for quantum computation from [6].

7. Related Work

Quantum Hoare Logic (QHL), as introduced by M. Ying in his work [21], was conceived with the aim of serving as a quantum counterpart to Hoare Logic. When examining the logical aspects, BDQL, in contrast to QHL, possesses the capability to express more foundational elements within quantum programs. Specifically, QHL faces limitations when dealing with the $\mathbf{if} \cdots \mathbf{fi}$ statement, which represents non-deterministic measurements, as it cannot be further divided. In contrast, BDQL can explicitly express its non-deterministic characteristic by employing the choice operator $\cup$. Additionally, it's worth noting that QHL lacks the inclusion of a test operator in its syntax and does not support probability in its formulas.

In our paper, we used Maude as our implementation language. In contrast, the work by [22] utilized PRISM, a probabilistic symbolic model checker, to verify quantum protocols. Unlike our approach, [22] requires the task of enumerating states and precomputing state transitions, which are then encoded into a specification in PRISM. Conversely, our method doesn't necessitate this state enumeration and precomputing state transitions because we formalize both the quantum computation and the semantics of PDQL using equations. The verification problem is carried out automatically through an equational simplification process within Maude. Moreover, they only support a small number of qubits, saying less than five qubits. As demonstrated, we can verify Quantum Network Coding with 14 qubits within a few moments, demonstrating the scalability of our approach.

8. Conclusion

We have extended Basic Dynamic Quantum Logic (BDQL) to Probabilistic Dynamic Quantum Logic (PDQL) to verify probabilistic quantum programs by introducing the probabilistic operator $\mathbf{P}^{\geq r}$ and some others in the formulas of PDQL. We have also developed a support tool in Maude to automate the entire verification process. Several quantum programs have been verified automatically using the support tool, demonstrating the usefulness of PDQL and our support tool. As one piece of our future work, we would like to conduct more case studies where the probabilistic properties are realistically expressed, such as Quantum Search Algorithm and Quantum Leader Election Protocol.

Acknowledgments

The research was supported by JAIST Research Grant for Fundamental Research. The research of the first author was supported by JSPS KAKENHI Grant Number JP23K19959. The research of the first and the third authors was supported by JST SICORP Grant Number JPMJSC20C2. The research of the second author was supported by Grant-in-Aid for JSPS Research Fellow Grant Number JP22KJ1483.

References

[1] P. Shor, Algorithms for quantum computation: discrete logarithms and factoring, in: Proceedings 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 124–134. doi:10.1109/SFCS.1994.365700.

[2] D. Harel, D. Kozen, J. Tiuryn, Dynamic Logic, MIT Press, 2000.

[3] A. Baltag, S. Smets, Reasoning about quantum information: An overview of quantum dynamic logic, Applied Sciences 12 (2022). URL: https://www.mdpi.com/2076-3417/12/9/4458. doi:10.3390/app12094458.

[4] T. Takagi, C. M. Do, K. Ogata, Automated quantum program verification in a dynamic quantum logic (to appear), in: DaLí: Dynamic Logic – New trends and applications, 2023.

[5] M. Clavel, et al., All About Maude, volume 4350 of *Lecture Notes in Computer Science*, Springer, 2007. doi:10.1007/978-3-540-71999-1.

[6] C. M. Do, K. Ogata, Symbolic model checking quantum circuits in maude, in: The 35th International Conference on Software Engineering and Knowledge Engineering, SEKE 2023, 2023, pp. 103–108. doi:10.18293/SEKE2023-014.

[7] C. H. Bennett, S. J. Wiesner, Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states, Phys. Rev. Lett. 69 (1992) 2881–2884. doi:10.1103/PhysRevLett.69.2881.

[8] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, W. K. Wootters, Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels, Phys. Rev. Lett. 70 (1993) 1895–1899. doi:10.1103/PhysRevLett.70.1895.

[9] M. Hillery, V. Bužek, A. Berthiaume, Quantum secret sharing, Phys. Rev. A 59 (1999) 1829–1834. doi:10.1103/PhysRevA.59.1829.

[10] M. Żukowski, A. Zeilinger, M. A. Horne, A. K. Ekert, "Event-ready-detectors" Bell experiment via entanglement swapping, Phys. Rev. Lett. 71 (1993) 4287–4290. doi:10.1103/PhysRevLett.71.4287.

[11] S.-T. Cheng, C.-Y. Wang, M.-H. Tao, Quantum communication for wireless wide-area networks, IEEE Journal on Selected Areas in Communications 23 (2005) 1424–1432. doi:10.1109/JSAC.2005.851157.

[12] S. Hassanpour, M. Houshmand, Bidirectional quantum teleportation and secure direct communication via entanglement swapping, 2014. doi:10.48550/arXiv.1411.0206. arXiv:1411.0206.

[13] D. Gottesman, I. L. Chuang, Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations, Nature 402 (1999) 390–393. doi:10.1038/46503.

[14] G. Rigolin, Quantum teleportation of an arbitrary two-qubit state and its relation to multipartite entanglement, Phys. Rev. A 71 (2005) 032303. URL: https://link.aps.org/doi/10.1103/PhysRevA.71.032303. doi:10.1103/PhysRevA.71.032303.

[15] T. Satoh, F. L. Gall, H. Imai, Quantum network coding for quantum repeaters, Physical Review A 86 (2012). doi:10.1103/physreva.86.032331.

[16] G. Birkhoff, J. von Neumann, The logic of quantum mechanics, Annals of mathematics 57 (1936) 823–843. doi:10.2307/1968621.

[17] M. Rédei, Quantum logic in algebraic approach, volume 91 of *Fundamental Theories of*

Physics, Springer, 1998. doi:10.1007/978-94-015-9026-6.

[18] A. Baltag, S. Smets, LQP: the dynamic logic of quantum information, Mathematical structures in computer science 16 (2006) 491–525. doi:10.1017/S0960129506005299.

[19] M. A. Nielsen, I. L. Chuang, Quantum Computation and Quantum Information: 10th Anniversary Edition, Cambridge University Press, 2011.

[20] J. Meseguer, Twenty years of rewriting logic, The Journal of Logic and Algebraic Programming 81 (2012) 721–781. doi:10.1016/j.jlap.2012.06.003.

[21] M. Ying, Floyd–hoare logic for quantum programs, ACM Transactions on Programming Languages and Systems (TOPLAS) 33 (2012) 1–49. doi:10.1145/2049706.2049708.

[22] S. Gay, R. Nagarajan, N. Papanikolaou, Probabilistic model–checking of quantum protocols, arXiv preprint quant-ph/0504007 (2005). doi:10.48550/arXiv.quant-ph/0504007.

Symbolic Model Checking Quantum Circuits With Density Operators in Maude

Canh Minh Do*, Kazuhiro Ogata

Japan Advanced Institute of Science and Technology, 1-8 Asahidai, Nomi, Japan

Abstract

We proposed a symbolic approach to model checking quantum circuits using a set of laws from quantum mechanics and basic matrix operations with Dirac notation and used Maude, a high-level specification/programming language based on rewriting logic, to implement our symbolic approach. However, it only supports the formalization of pure states but not mixed states of quantum systems. This paper extends our current symbolic approach to deal with mixed states of quantum systems by using density operators for their representations. Once a quantum circuit is formalized by our proposed way with an initial state and a desired property expressed in Linear Temporal Logic (LTL), we use a built-in Maude LTL model checker to verify whether the quantum circuit enjoys the desired property from the initial state automatically. As case studies, we successfully verify several quantum communication protocols: Superdense Coding, Quantum Teleportation, Quantum Secret Sharing, and Entanglement Swapping.

Keywords

Mixed States, Density Operators, Quantum circuits, Dirac notation, Symbolic Model Checking, Maude

1. Introduction

Quantum computing represents a new frontier in computation, offering the potential to solve hard problems, which were previously considered intractable for the current computing technologies. For example, Shore's fast algorithm [1] can break the security of modern cryptographic systems relying on discrete logarithms and factoring in the future once practical quantum computers are available. To design and implement such quantum algorithms, quantum circuits are often used as a model of quantum computation. Unlike classical circuits, quantum circuits manipulate qubits using quantum operations (e.g., quantum gates). Because quantum computation is counter-intuitive and distinct from classical computing due to different principles, such as superposition, entanglement, and measurement, it is challenging to design and implement quantum algorithms (or quantum circuits) accurately. Therefore, it is crucial to verify that quantum circuits enjoy some desired properties in preparing for the quantum era.

Our research group proposed a symbolic approach [2] to model checking quantum circuits using a set of laws from quantum mechanics and basic matrix operations with Dirac notation [3] and used Maude [4], a high-level specification/programming language based on rewriting

The 2nd International Workshop on Formal Analysis and Verification of Post-Quantum Cryptographic Protocols, November 11, 2023, Brisbane, Australia

*Corresponding author.

canhdo@jaist.ac.jp (C. M. Do); ogata@jaist.ac.jp (K. Ogata)

0000-0002-1601-4584 (C. M. Do); 0000-0002-4441-3259 (K. Ogata)

logic [5], to implement our symbolic approach. However, it only supports the formalization of pure states but not mixed states of quantum systems. In many practical situations, a quantum system is not a single, well-defined state but a statistical mixture of multiple pure states. Therefore, it requires the formalization of mixed states to describe the probabilities associated with each pure state. This paper extends our current symbolic approach [2] to deal with mixed states of quantum systems by using density operators for their representations. Once a quantum circuit is formalized by our proposed way with an initial state and a desired property expressed in Linear Temporal Logic (LTL), we use the built-in Maude LTL model checker to automatically verify whether the quantum circuit enjoys the desired property from the initial state. As case studies, we successfully verify several quantum communication protocols: Superdense Coding [6], Quantum Teleportation [7], Quantum Secret Sharing [8], and Entanglement Swapping [9]. This demonstrates the usefulness of our symbolic model checking quantum circuits with density operators in Maude. The support tool and case studies are publicly available at https://github.com/canhminhdo/QTC-Maude under the `mixed-states` folder.

The rest of the paper is organized as follows: Sect. 2 provides basic quantum mechanics, symbolic reasoning for quantum computation based on Dirac notation, and Kripke structure; Sect. 3 describes the formalization of qubits, quantum gates, and quantum circuits; Sect. 4 demonstrates how to use our approach to model checking for Quantum Teleportation; Sect. 5 shows the experimental results with our support tool; Sect. 6 presents some existing work; and Sect. 7 concludes the paper with some pieces of future work.

2. Preliminaries

Firstly, we describe some basic notations from quantum mechanics based on linear algebra (refer to [10] for more details). Secondly, we describe symbolic reasoning [2, 11] to reason about quantum computation based on Dirac notation. Lastly, we describe Kripke structures to formalize quantum systems.

2.1. Basic Quantum Mechanics

In classical computing, the fundamental unit of information is a bit whose value is either 0 or 1. In quantum computing, the counterpart is a *quantum bit* or *qubit*, which has two basis states, conventionally written in Dirac notation [3] as $|\mathbf{0}\rangle$ and $|\mathbf{1}\rangle$, corresponding to one-bit classical values, whose values are two column vectors $\begin{pmatrix}1\\0\end{pmatrix}$ and $\begin{pmatrix}0\\1\end{pmatrix}$, respectively. In quantum theory, a general state of a quantum system is a superposition or linear combination of basis states. A single qubit has state $|\psi\rangle = \alpha\,|\mathbf{0}\rangle + \beta\,|\mathbf{1}\rangle$, where α and β are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$. States can be represented by column complex vectors as follows: $|\psi\rangle = \begin{pmatrix}\alpha\\\beta\end{pmatrix} = \alpha\,|\mathbf{0}\rangle + \beta\,|\mathbf{1}\rangle$, where $\{|\mathbf{0}\rangle, |\mathbf{1}\rangle\}$ forms an orthonormal basis of the 2D complex vector space. Formally, a quantum state is a unit vector in a Hilbert space $\mathcal{H}$, which is equipped with an inner product satisfying some axioms.

The basis $\{|\mathbf{0}\rangle, |\mathbf{1}\rangle\}$ is called as the *standard* basis. Besides, we have some other bases of interest, such as *diagonal* (or *dual*, or *Hadamard*) basis consisting of the following vectors:

$$|+\rangle = \tfrac{1}{\sqrt{2}}(|\mathbf{0}\rangle + |\mathbf{1}\rangle) \text{ and } |-\rangle = \tfrac{1}{\sqrt{2}}(|\mathbf{0}\rangle - |\mathbf{1}\rangle)$$

The evolution of a closed quantum system can be performed by a unitary transformation. If the state of a qubit is represented by a column vector, then a unitary transformation $\boldsymbol{U}$ can be represented by a complex-value matrix such that $\boldsymbol{U}\boldsymbol{U}^\dagger = \boldsymbol{U}^\dagger\boldsymbol{U} = \boldsymbol{I}$ or $\boldsymbol{U}^\dagger = \boldsymbol{U}^{-1}$, where $\boldsymbol{U}^\dagger$ is the conjugate transpose of $\boldsymbol{U}$. The trace of $\boldsymbol{U}$ is defined as $tr(\boldsymbol{U}) = \sum_i \langle\phi_i|\boldsymbol{U}|\phi_i\rangle$ for some given orthonormal basis $\{|\phi_i\rangle\}$ of $\mathcal{H}$. $\boldsymbol{U}$ acts on the Hilbert space $\mathcal{H}$ transforming a state $|\psi\rangle$ to a state $|\psi'\rangle$ by a matrix multiplication such that $|\psi'\rangle = \boldsymbol{U}|\psi\rangle$. There are some common quantum gates: the Hadamard gate $\boldsymbol{H}$, the identity gate $\boldsymbol{I}$, the Pauli gates $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$, and the controlled-NOT gate $\boldsymbol{CX}$. Note that the $\boldsymbol{CX}$ gate performs on two qubits, while the remaining gates perform on a single qubit. Their matrix representations are as follows:

$$\boldsymbol{I}_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \boldsymbol{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad \boldsymbol{Y} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix},$$

$$\boldsymbol{Z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \qquad \boldsymbol{H} = \tfrac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad \boldsymbol{CX} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

where i is the imaginary unit. For example, the Hadamard gate on a single qubit performs the mapping $|\mathbf{0}\rangle \mapsto \frac{1}{\sqrt{2}}(|\mathbf{0}\rangle + |\mathbf{1}\rangle)$ and $|\mathbf{1}\rangle \mapsto \frac{1}{\sqrt{2}}(|\mathbf{0}\rangle - |\mathbf{1}\rangle)$. The controlled-NOT gate on pairs of qubits performs the mapping $|\mathbf{00}\rangle \mapsto |\mathbf{00}\rangle, |\mathbf{01}\rangle \mapsto |\mathbf{01}\rangle, |\mathbf{10}\rangle \mapsto |\mathbf{11}\rangle, |\mathbf{11}\rangle \mapsto |\mathbf{10}\rangle$, which can be understood as inverting the second qubit (referred to as the *target*) if and only if the first qubit (referred to as the *control*) is one.

A quantum measurement is described as a collection $\{\boldsymbol{M}_m\}$ of measurement operators, where the indices m refer to the measurement outcomes. It is required that the measurement operators satisfy $\sum_m \boldsymbol{M}_m^\dagger \boldsymbol{M}_m = \boldsymbol{I}_\mathcal{H}$. If the state of a quantum system is $|\psi\rangle$ before the measurement, then the probability for the result m is as follows:

$$p(m) = \langle\psi|\, \boldsymbol{M}_m^\dagger \boldsymbol{M}_m\, |\psi\rangle,$$

and the state of the quantum system after the measurement is $\frac{\boldsymbol{M}_m|\psi\rangle}{\sqrt{p(m)}}$ provided that $p(m) > 0$. For example, if a qubit is in state $\alpha|\mathbf{0}\rangle + \beta|\mathbf{1}\rangle$ and measuring with $\{\boldsymbol{M}_0, \boldsymbol{M}_1\}$ operators, we have the result 0 with probability $|\alpha|^2$ at the post-measurement state $|\mathbf{0}\rangle$ and the result 1 with probability $|\beta|^2$ at the post-measurement state $|\mathbf{1}\rangle$, where $\boldsymbol{M}_0 = |\mathbf{0}\rangle \times \langle\mathbf{0}|$ and $\boldsymbol{M}_1 = |\mathbf{1}\rangle \times \langle\mathbf{1}|$.

For multiple qubits, we use the tensor product of Hilbert spaces. Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be two Hilbert spaces. Their tensor product $\mathcal{H}_1 \otimes \mathcal{H}_2$ is defined as a vector space consisting of linear combinations of the vectors $|\psi_1\psi_2\rangle = |\psi_1\rangle\,|\psi_2\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$, where $|\psi_1\rangle \in \mathcal{H}_1$ and $|\psi_2\rangle \in \mathcal{H}_2$. Systems of two or more qubits may be in *entangled* states, meaning that states of qubits are

correlated and inseparable. Entanglement shows that an entangled state of two qubits cannot be expressed as a tensor product of single-qubit states. We can use $\boldsymbol{H}$ and $\boldsymbol{CX}$ gates to create entangled states as follows: $\boldsymbol{CX}\left((\boldsymbol{H}\otimes\boldsymbol{I})\left|\mathbf{00}\right\rangle\right)=\frac{1}{\sqrt{2}}(\left|\mathbf{00}\right\rangle+\left|\mathbf{11}\right\rangle)$.

A pure quantum state can be represented by a ket vector $|\psi\rangle$, while a mixed state represents the probabilistic mixtures of pure states and can be described by a density operator. Given $\{(p_i,|\psi_i\rangle)\}$ be an ensemble of pure states $|\psi_i\rangle$, where $p_i\geq 0$ and $\sum_i p_i=1$, $\rho=\sum_i p_i\left|\psi_i\right\rangle\langle\psi_i|$ is the density operator representing the mixed state. $\boldsymbol{U}$ transforms a mixed state ρ to a mixed state ρ' by a matrix multiplication such that $\rho'=\boldsymbol{U}\rho\boldsymbol{U}^\dagger$. For the quantum measurement $\{\boldsymbol{M}_m\}$ above, if the mixed state of a quantum system is ρ before the measurement, then the probability for the result m is as follows:

$$p(m)=tr(\boldsymbol{M}_m^\dagger\boldsymbol{M}_m\rho),$$

and the mixed state after the measurement is $\frac{\boldsymbol{M}_m\rho\boldsymbol{M}_m^\dagger}{p(m)}$ provided that $p(m)>0$.

2.2. Symbolic Reasoning

We proposed symbolic reasoning [2, 11] based on Dirac notation with scalars using a set of laws from quantum mechanics and basic matrix operations for reasoning about quantum computation and developed a support tool in Maude to automate the reasoning. This section briefly describes terms used in our symbolic reasoning and laws used to reduce terms. The reader is recommended to refer to [2] for more details.

2.2.1. Terms

Terms are built from scalars and basis vectors with some constructors.

- Scalars are complex numbers. We extend rational numbers supported in Maude to deal with complex numbers. Some constructors for scalars, such as multiplication, fraction, addition, conjugation, absolute, power, and square root are formalized, but we do not mention them here to make the paper concise.
- Basis vectors are the standard basis written in Dirac notation as $\left|\mathbf{0}\right\rangle$ and $\left|\mathbf{1}\right\rangle$.
- Constructors for matrices consist of scalar multiplication of matrices $\cdot$, matrix product $\times$, matrix addition $+$, tensor product $\otimes$, and the conjugate transpose $\boldsymbol{A}^\dagger$ of a matrix $\boldsymbol{A}$.

In Dirac notation, $\langle\mathbf{0}|$ is the dual of $\left|\mathbf{0}\right\rangle$ such that $\langle\mathbf{0}|^\dagger=\left|\mathbf{0}\right\rangle$ and $\left|\mathbf{0}\right\rangle^\dagger=\langle\mathbf{0}|$; similarly for $\langle\mathbf{1}|$. The terms $\left|j\right\rangle\times\langle k|$ and the inner product of $\left|j\right\rangle$ and $\left|k\right\rangle$ may be written shortly as $\left|j\right\rangle\langle k|$ and $\langle j|k\rangle$ for any $j,k\in\{0,1\}$. By using these notations with the laws below, we can intuitively explain how quantum operations work.

2.2.2. Laws

Table 1 presents a set of laws derived from the properties of quantum mechanics and basic matrix operations. Because $\left|\mathbf{0}\right\rangle$ and $\left|\mathbf{1}\right\rangle$ can be viewed as 2×1 matrices, then the laws actually describe matrix calculations with Dirac notation, zero and identity matrices, and scalars. These

Table 1
A set of laws used for symbolic reasoning

No.	Law
L1	$\langle \mathbf{0}\|\mathbf{0}\rangle = \langle \mathbf{1}\|\mathbf{1}\rangle = 1, \langle \mathbf{1}\|\mathbf{1}\rangle = \langle \mathbf{0}\|\mathbf{1}\rangle = 0$
L2	Associativity of $\times, +, \otimes$ and Commutativity of $+$
L3	$0 \cdot \boldsymbol{A}_{m\times n} = \mathbf{O}_{m\times n},\ c \cdot \mathbf{O} = \mathbf{O},\ 1 \cdot \boldsymbol{A} = \boldsymbol{A}$
L4	$c \cdot (\boldsymbol{A} + \boldsymbol{B}) = c \cdot \boldsymbol{A} + c \cdot \boldsymbol{B}$
L5	$c_1 \cdot \boldsymbol{A} + c_2 \cdot \boldsymbol{A} = (c_1 + c_2) \cdot \boldsymbol{A}$
L6	$c_1 \cdot (c_2 \cdot \boldsymbol{A}) = (c_1 \cdot c_2) \cdot \boldsymbol{A}$
L7	$(c_1 \cdot \boldsymbol{A}) \times (c_2 \cdot \boldsymbol{B}) = (c_1 \cdot c_2) \cdot (\boldsymbol{A} \times \boldsymbol{B})$
L8	$\boldsymbol{A} \times (c \cdot \boldsymbol{B}) = (c \cdot \boldsymbol{A}) \times \boldsymbol{B} = c \cdot (\boldsymbol{A} \times \boldsymbol{B})$
L9	$\boldsymbol{A} \otimes (c \cdot \boldsymbol{B}) = (c \cdot \boldsymbol{A}) \otimes \boldsymbol{B} = c \cdot (\boldsymbol{A} \otimes \boldsymbol{B})$
L10	$\mathbf{O}_{m\times n} \times \boldsymbol{A}_{n\times p} = \boldsymbol{A}_{m\times n} \times \mathbf{O}_{n\times p} = \mathbf{O}_{m\times p}$
L11	$\boldsymbol{I}_m \times \boldsymbol{A}_{m\times n} = \boldsymbol{A}_{m\times n} \times \boldsymbol{I}_n = \boldsymbol{A}_{m\times n}$
L12	$\boldsymbol{A} + \mathbf{O} = \mathbf{O} + \boldsymbol{A} = \mathbf{O}$
L13	$\mathbf{O}_{m\times n} \otimes \boldsymbol{A}_{p\times q} = \boldsymbol{A}_{p\times q} \otimes \mathbf{O}_{m\times n} = \mathbf{O}_{mp\times nq}$
L14	$\boldsymbol{A} \times (\boldsymbol{B} + \boldsymbol{C}) = \boldsymbol{A} \times \boldsymbol{B} + \boldsymbol{A} \times \boldsymbol{C}$
L15	$(\boldsymbol{A} + \boldsymbol{B}) \times \boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{C} + \boldsymbol{B} \times \boldsymbol{C}$
L16	$(\boldsymbol{A} \otimes \boldsymbol{B}) \times (\boldsymbol{C} \otimes \boldsymbol{D}) = (\boldsymbol{A} \times \boldsymbol{C}) \otimes (\boldsymbol{B} \times \boldsymbol{D})$
L17	$\boldsymbol{A} \otimes (\boldsymbol{B} + \boldsymbol{C}) = \boldsymbol{A} \otimes \boldsymbol{B} + \boldsymbol{A} \otimes \boldsymbol{C}$
L18	$(\boldsymbol{A} + \boldsymbol{B}) \otimes \boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{C} + \boldsymbol{B} \otimes \boldsymbol{C}$
L19	$(c \cdot \boldsymbol{A})^\dagger = c^* \cdot \boldsymbol{A}^\dagger,\ (\boldsymbol{A} \times \boldsymbol{B})^\dagger = \boldsymbol{B}^\dagger \times \boldsymbol{A}^\dagger$
L20	$(\boldsymbol{A} + \boldsymbol{B})^\dagger = \boldsymbol{A}^\dagger + \boldsymbol{B}^\dagger,\ (\boldsymbol{A} \otimes \boldsymbol{B})^\dagger = \boldsymbol{A}^\dagger \otimes \boldsymbol{B}^\dagger$
L21	${\boldsymbol{I}_m}^\dagger = \boldsymbol{I}_m, \boldsymbol{O}_{m\times n}^\dagger = \boldsymbol{O}_{n\times m}, (\boldsymbol{A}^\dagger)^\dagger = \boldsymbol{A}$
L22	$\|\mathbf{0}\rangle^\dagger = \langle \mathbf{0}\|,\ \langle \mathbf{0}\|^\dagger = \|\mathbf{0}\rangle,\ \|\mathbf{1}\rangle^\dagger = \langle \mathbf{1}\|,\ \langle \mathbf{1}\|^\dagger = \|\mathbf{1}\rangle$

laws are described by equations in Maude and are used to automatically reduce terms until no more matrix operation is applicable. Some laws dedicated to simplifying the expressions about complex numbers are also formalized in Maude by means of equations.

2.3. Kripke Structures

A Kripke structure K is $\langle S, I, T, A, L\rangle$ [12], where S is a set of states, $I \subseteq S$ is the set of initial states, $T \subseteq S \times S$ is a left-total binary relation over S, A is a set of atomic propositions and L is a labeling function whose type is $S \to 2^A$. Each element $(s, s') \in T$ is called a state transition from s to s' and T may be called the state transitions (with respect to K). For a state $s \in S$, $L(s)$ is the set of atomic propositions that hold in s. A path π is an infinite sequence $s_0, \ldots, s_i, s_{i+1}, \ldots$ such that $s_i \in S$ and $(s_i, s_{i+1}) \in T$ for each i. We use the following notations for paths: $\pi^i \triangleq s_i, s_{i+1}, \ldots$, $\pi_i \triangleq s_0, \ldots, s_i, s_i, s_i, \ldots$, $\pi(i) \triangleq s_i$, where $\triangleq$ is used as "be defined as." π^i is obtained by deleting the first i states $s_0, s_1, \ldots, s_{i-1}$ from π. π_i is obtained

by taking the first $i+1$ states $s_0, s_1, \ldots, s_{i-1}, s_i$ and adding s_i unboundedly many times at the end. $\pi(i)$ is the ith state s_i. Let $\mathcal{P}$ be the set of all paths. π is called a computation if $\pi(0) \in I$. Let $\mathcal{C}$ be the set of all computations.

The syntax of a formula φ in LTL for K is as follows:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid \varphi \,\mathcal{U}\, \varphi$$

where $p \in A$. Let $\mathcal{F}$ be the set of all formulas in LTL for K. An arbitrary path $\pi \in \mathcal{P}$ of K and an arbitrary LTL formula $\varphi \in \mathcal{F}$ of K, $K, \pi \models \varphi$ is inductively defined as follows:

- $K, \pi \models \top$
- $K, \pi \models p$ iff $p \in L(\pi(0))$
- $K, \pi \models \neg\varphi_1$ iff $K, \pi \not\models \varphi_1$
- $K, \pi \models \varphi_1 \wedge \varphi_2$ iff $K, \pi \models \varphi_1$ and $K, \pi \models \varphi_2$
- $K, \pi \models \bigcirc \varphi_1$ iff $K, \pi^1 \models \varphi_1$
- $K, \pi \models \varphi_1 \,\mathcal{U}\, \varphi_2$ iff there exists a natural number i such that $K, \pi^i \models \varphi_2$ and for all natural numbers $j < i$, $K, \pi^j \models \varphi_1$

where φ_1 and φ_2 are LTL formulas. Then, $K \models \varphi$ iff $K, \pi \models \varphi$ for each computation $\pi \in \mathcal{C}$ of K. $\bigcirc$ and $\mathcal{U}$ are called the next temporal connective and the until temporal connective, respectively. We define $\Diamond \varphi \triangleq \top \,\mathcal{U}\, \varphi$, where $\Diamond$ is called the eventual temporal connective. In this paper, we use LTL formulas to express desired properties under verification for our case studies.

In this paper, a state is expressed as a braced associative-commutative (AC) collection of name-value pairs, where a name may have parameters. The order of elements is not relevant in AC collections, such as sets. AC collections are called soups, and name-value pairs are called observable components. That is, a state is expressed as a braced soup of observable components. The juxtaposition operator is used as the constructor of soups. Let oc_1, oc_2, oc_3 be observable components, and then $oc_1\ oc_2\ oc_3$ is the soup of those three observable components. Since the order is irrelevant because of AC, $oc_1\ oc_2\ oc_3$ is the same as some others, such as $oc_3\ oc_2\ oc_1$. A state is expressed as $\{oc_1\ oc_2\ oc_3\}$. In this paper, rewrite rules are used to specify state transitions. Concretely, we use Maude [4], a programming/specification language based on rewriting logic. Maude makes it possible to specify complex systems flexibly and is also equipped with a built-in LTL model checker to conduct model checking experiments.

3. Formal Specification

This section summarizes how we formalize qubits, quantum gates, measurements, and then quantum circuits in [2] and describes how we extend it to deal with mixed states.

3.1. Formalization of Qubits, Gates, and Measurements

Pure states of qubits are represented in our formalization as the linear combination of tensor product of the standard basis in Dirac notation with scalars. This representation is also adopted

for quantum gates. In our work, we exclusively focus on binary projective measurements conducted on the standard basis. This implies that our measurement operators consist of two elements, denoted as $\boldsymbol{M}_0$ and $\boldsymbol{M}_1$. To formalize a mixed state, we prepare an ensemble of pure states and calculate its corresponding density operator as described in Section 2.

3.2. A Formalization of Quantum Circuits

Our approach to formalizing quantum circuits begins by representing them as a sequence comprising various actions, including quantum gates, measurements, qubit initializations, and other operations, as depicted in Figure 1. Following this, we establish Kripke structures tailored to quantum circuits to conduct model checking that quantum circuits enjoy desired properties.

3.2.1. Elements of Quantum Circuits

A whole quantum state is formalized as a density operator representing a mixed state. In our previous work [2], a whole quantum state is formalized as a pure state, which is a linear combination of the tensor product of basic vectors with Dirac notation and complex numbers.

Classical bits are formalized the same as in our previous work [2]. They are represented as a mapping from circuit indices to Boolean values. Each entry in this mapping takes the form of $(i \mapsto b)$, indicating that the value of the classical bit stored at position c_i is b, with b taking on values of either 0 or 1.

A sequence comprising *quantum gates, measurements*, and *conditional gates* in a quantum circuit is formalized as a list of actions. Each action can take one of the following forms:

- `I`(i) applies the $\boldsymbol{I}$ gate on qubit at index i,
- `X`(i) applies the $\boldsymbol{X}$ gate on qubit at index i,
- `Y`(i) applies the $\boldsymbol{Y}$ gate on qubit at index i,
- `Z`(i) applies the $\boldsymbol{Z}$ gate on qubit at index i,
- `H`(i) applies the $\boldsymbol{H}$ gate on qubit at index i,
- `CX`(i, j) applies the $\boldsymbol{CX}$ gate on qubits at indices i and j,
- `CY`(i, j) applies the $\boldsymbol{CY}$ gate on qubits at indices i and j,
- `CZ`(i, j) applies the $\boldsymbol{CZ}$ gate on qubits at indices i and j,
- `SWAP`(i, j) applies the $\boldsymbol{SWAP}$ gate on qubits at indices i and j,
- `CCX`(i, j, k) applies the $\boldsymbol{CCX}$ gate on qubits at indices i, j and k,
- `CCZ`(i, j, k) applies the $\boldsymbol{CCX}$ gate on qubits at indices i, j and k,
- `CSWAP`(i, j, k) applies the $\boldsymbol{CSWAP}$ gate on qubits at indices i, j and k,
- `M`(i) measures q_i with the standard basis,
- `c[`i`] == `b` ? AL` checks if the classical bit at c_i equals b, then a list `AL` of actions is executed.

All actions except the two last actions representing measurements and conditional actions are called basic actions. In this paper, we support more quantum gates than in our previous work [2], including $\boldsymbol{S}$, $\boldsymbol{T}$, $\boldsymbol{CY}$, $\boldsymbol{CZ}$, $\boldsymbol{SWAP}$, $\boldsymbol{CCY}$, $\boldsymbol{CCZ}$, and $\boldsymbol{CSWAP}$ gates.

3.2.2. Kripke Structures of Quantum Circuits

Let K be the Kripke structure formalizing a quantum circuit. In our formalization, we define five distinct observable components as follows:

- `(mState: ` ms`)` denotes the mixed quantum state ms.
- `(#qubits: ` n`)` denotes the number of qubits n.
- `(bits: ` bm`)` denotes the classical bits obtained from measurements and stored in a bit map bm.
- `(prob: ` p`)` denotes the probability p at the current quantum state.
- `(actions: ` al`)` denotes the action list al, guiding us on how the circuit works.
- `(isEnd: ` b`)` denotes termination with Boolean flag b.

Each state in S is expressed as $\{obs\}$, where obs is a soup of those five distinct observable components. The `mState` and `#qubits` observable components are newly added to deal with mixed states compared to our previous work [2].

T now consists of six rewrite rules in our formalization. Let `OCs` be a Maude variable of observable component soups, `MS` and `MS'` be Maude variables of whole quantum states, `BM` be a Maude variable of bit maps, `Prob` and `Prob'` be Maude variables of scalars, `AL` and `AL'` be Maude variables of action lists, `B` be a Maude variable of Boolean values, and `N`, `N'`, `N1`, and `N2` are Maude variables of natural numbers.

The first rewrite rule is as follows:

```
crl [U] : {(mState: MS) (actions: (A AL)) (#qubits: N) OCs}
=> {(mState: MS') (actions: AL) (#qubits: N) OCs}
if isBasicAction(A) /\ MS' := unitary(MS, A, N) .
```

The rule `U` simulates unitary transformation on the whole quantum state in `mState` observable component if its basic action appears in `actions` observable component.

The next two rewrite rules are as follows:

```
crl [M0] :
{(mState: MS) (actions: (M(N') AL)) (prob: Prob) (bits: BM) (#qubits: N) OCs}
=> {(mState: MS') (actions: AL) (prob: (Prob .* Prob')) (bits: insert(N', 0, BM))
    (#qubits: N) OCs}
if {mState: MS', prob: Prob'} := measure(MS, N, P0, N') .

crl [M1] :
{(mState: MS) (actions: (M(N') AL)) (prob: Prob) (bits: BM) (#qubits: N) OCs}
=> {(mState: MS') (actions: AL) (prob: (Prob .* Prob')) (bits: insert(N', 1, BM))
    (#qubits: N) OCs}
if {mState: MS', prob: Prob'} := measure(MS, N, P1, N') .
```

where `P0` and `P1` are Maude constants of matrices representing the measurement operators $\boldsymbol{M}_0$ and $\boldsymbol{M}_1$, respectively. Therefore, the rules `M0` and `M1` govern the process of measuring a qubit at index `N'` with the measurement operators $\boldsymbol{M}_0$ and $\boldsymbol{M}_1$, respectively. In these rules, the classical outcomes are stored accordingly into the bit map in `bits` observable component;

the probabilities and the post-measurement mixed states are updated in `prob` and `mState` observable components, respectively. It is important to note that these two rules introduce non-deterministic probabilistic transitions when measuring a single qubit.

The next rewrite rule is identical to the one presented in our previous work [2]. It outlines the processes of conditionally executing the next actions based on classical bits obtained from measurements if applicable.

```
rl [cif] :
{(qstate: Q) (bits: ((N |-> N1),BM)) (actions: ((c[N] == N2 ? AL') AL)) OCs}
=> {(qstate: Q) (bits: ((N |-> N1), BM))
    (actions: ((if (N1 == N2) then AL' else nil fi) AL)) OCs} .
```

This rule says that if `c[N] == N2 ? AL'` exists in the action list and the classical bit `N1` at index `N` equals the conditional value `N2`, then the action list `AL'` is added at the beginning of the action list `AL` in `actions` observable component. This means that `AL'` will be executed next. Otherwise, it is simply ignored and `AL` remains unchanged.

The last two rules are the same as in our previous work [2] as follows:

```
rl [end] : {(actions: nil) (isEnd: false) OCs}
=> {(actions: nil) (isEnd: true) OCs} .

rl [stutter]: {(isEnd: true) OCs} => {(isEnd: true) OCs} .
```

The rule `end` indicates termination when the action list is empty, denoted as `nil`. On the other hand, the rule `stutter` is to make T total when `isEnd` observable component is true.

4. A Case Study: Quantum Teleportation

For the sake of simplicity, this section only demonstrates how to use our symbolic approach with density operators to conduct model checking for Quantum Teleportation [7]. Meanwhile, other communication protocols are similar and the full specifications of all quantum communication protocols concerned in this paper are publicly available at https://github.com/canhminhdo/QTC-Maude under the `mixed-states` folder.

4.1. Introduction

Quantum Teleportation, as described in [7], leverages the unique properties of entanglement in quantum mechanics to transmit an unknown quantum state $|\psi\rangle$ from Alice to Bob, utilizing only three qubits and two classical bits. This protocol holds significant importance because of the no-cloning theorem [13], which prohibits the exact copy of an arbitrary unknown quantum state. As a result, the protocol becomes a crucial method for transmitting an arbitrary unknown quantum state from one source to another.

The circuit illustrated in Figure 1 presents how the protocol works. Alice manipulates on qubits q_0 and q_1, and Bob manipulates on qubit q_2 as follows:

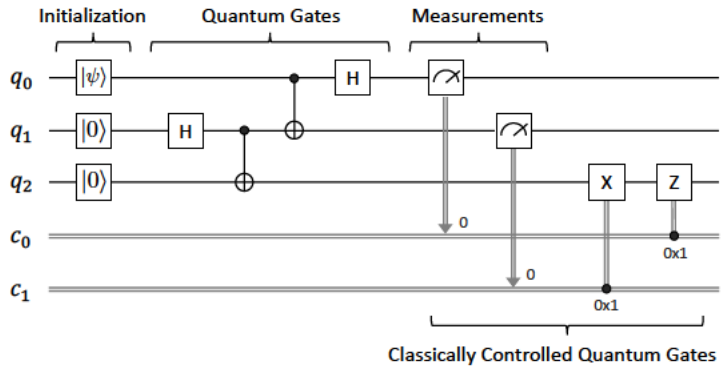


Figure 1: Quantum Teleportation

- *State preparation*: Initially, an unknown state $|\psi\rangle$ is prepared at qubit q_0, where $|\psi\rangle = \alpha|\mathbf{0}\rangle + \beta|\mathbf{1}\rangle$. The complex numbers α and β are such that $|\alpha|^2 + |\beta|^2 = 1$. Initially, qubits q_1 and q_2 are in the state $|\mathbf{0}\rangle$.
- *Gate operations*: A sequence of quantum gates is applied to manipulate the three qubits. This includes the single-qubit Hadamard $\boldsymbol{H}$ and two-qubit controlled-NOT $\boldsymbol{CX}$ gates. The following operations are performed: Alice and Bob share an entangled state by applying the $\boldsymbol{H}$ gate to qubit q_1 and the $\boldsymbol{CX}$ gate to qubits q_1 and q_2. Alice further manipulates qubits q_0 and q_1 by applying the $\boldsymbol{CX}$ gate to them, followed by the $\boldsymbol{H}$ gate to qubit q_0.
- *Measurement*: Qubits q_0 and q_1 are measured, yielding two classical outcomes (0 or 1) stored in c_0 and c_1, respectively.
- *Conditional gates*: Based on the classical bits in c_0 and c_1, conditional single-qubit $\boldsymbol{X}$ and $\boldsymbol{Z}$ gates are applied to qubit q_2. Specifically, the $\boldsymbol{X}$ gate is used if c_1 equals one, followed by the $\boldsymbol{Z}$ gate if c_0 equals one.

Finally, Bob will receive the original quantum state $|\psi\rangle$, while Alice will no longer have it. Our goal is to verify whether Alice can successfully transmit an arbitrary unknown quantum state to Bob at the end of this protocol by using our symbolic model checking approach with density operators.

4.2. Formalization of Quantum Teleportation

We can describe the circuit for Quantum Teleportation based on the actions formalized in Sect. 3 as follows:

```
H(1) CX(1, 2) CX(0, 1) H(0) M(0) M(1) (c[1] == 1 ? X(2)) (c[0] == 1 ? Z(2))
```

Let ES be an ensemble of pure states as follows:

```
{ (a . |0> + b . |1>) (x) |0>  (x) |0>, 1 }
```

where (x) denotes the tensor product; and a and b are Maude constants of scalars denoting arbitrary scalars such that $|a|^2 + |b|^2 = 1$. We suppose that the mixed state consists of only one pure state with a certain probability in the ensemble.

The set I of initial states for Quantum Teleportation includes only one initial state as follows:

```
{(isEnd: false)
(#qubits: findN(ES))
(mState: convert(ES)
(prob: 1)
(bits: empty)
(actions: H(1) CX(1, 2) CX(0, 1) H(0)
          M(0) M(1)
          c[1] == 1 ? X(2)
          c[0] == 1 ? Z(2))}
```

where `findN(_)` and `convert(_)` are two functions to calculate the number of qubits and the density operator of a mixed state given an ensemble. Initially, `isEnd` observable component is false, `prob` observable component is one, `mState` represent the corresponding density operator of the symbolic state as the input state of the protocol, `actions` observable component contains the action list describing how the protocol works.

4.3. Model Checking Quantum Teleportation

Let K and `init` be the Kripke structure and the initial state for Quantum Teleportation, respectively. In order to perform model checking on the Kripke structure K and verify that it satisfies the desired properties, we define the set of atomic propositions A and the labeling function L. A contains one atomic proposition, denoted as `isSuccess`. L is defined as follows:

```
eq {(isEnd: true) (mState: MS) (prob: Prob) (#qubits: N) OCs} |= isSuccess
= Prob > 0 implies
  tr[1]((tr[0](MS, N)), N) == (I (x) I (x) (PSI x (PSI)^+)) .
eq {OCs} |= PROP = false [owise] .
```

where `PSI` is the input state of the protocol being transferred, the function `tr[_](_,_)` take inputs as the index at which the information of the qubit is erased, the mixed state, and the number of qubits. The function will erase information of a subsystem from the mixed state at an index. Using this function we can retain the information of the qubit at the index 2 in the density operator. This function works as the partial trace over a quantum system [10].

The two equations say that `isSuccess` holds at a state if the state contains `(isEnd: true)`, `(mState: MS)`, `(prob: Prob)`, and `(#qbits: N)` such that the condition `tr[1]((tr[0](MS, N)), N) == (I (x) I (x) (PSI x (PSI)^+)` holds whenever `Prob > 0` holds, meaning that the qubit received by Bob at the end is equal to the qubit sent by Alice at the beginning with a non-zero probability by means of density operators. Notice that, the use of density operators to represent quantum states can eliminate the global phase when comparing two quantum states. Let `teleProp` be an LTL formula defined as `<> isSuccess`, where `<>` is the eventual temporal connective.

Table 2
Experimental results with pure states and mixed states for representing quantum states

Protocol	Qubits	States	Pure States		Mixed States	
			Rewrite Steps	Time	Rewrite Steps	Time
Superdense Coding	2	9	685	$\approx$ 0ms	2,088	2ms
Quantum Teleportation	3	27	4,340	3ms	29,095	30ms
Quantum Secret Sharing	4	65	16,449	9ms	211,831	519ms
Entanglement Swapping	4	33	6,930	4ms	56,193	40ms

We want to model check that $K = \langle S, I, T, A, L \rangle$ satisfies `teleProp` from the initial state `init` in Maude as follows:

```
red modelCheck(init, teleProp) .
```

No counterexample is found in just a few moments and so K satisfies `teleProp`. In other words, we successfully verify the correctness of Quantum Teleportation by using our symbolic model checking approach with density operators.

5. Experimental Results

We used an iMac that carries a 4 GHz microprocessor with eight cores and 32 GB memory RAM to conduct experiments in this section. As case studies, we conduct model checking experiments to verify the correctness of several quantum communication protocols with our approach in which both pure states and mixed states are used for representing quantum states as follows:

- Superdense Coding [6] for transmitting two classical bits using an entangled state,
- Quantum Teleportation [7] for teleporting an arbitrary pure state by sending two bits of classical information,
- Quantum Secret Sharing [8] for teleporting a pure state from a sender (Alice) to a receiver (Bob) with the help of a third party (Charlie),
- Entanglement Swapping [9] for creating a new entangled state,

Superdense Coding is the simplest one that uses only two qubits; Quantum Teleportation uses three qubits; Quantum Secret Sharing proposed relying on the mechanism of Quantum Teleportation uses four qubits; and Entanglement Swapping uses four qubits. Note that the properties being verified for both pure states and mixed states are identical in our experiments. Our support tool and case studies are publicly available at https://github.com/canhminhdo/QTC-Maude under the `mixed-states` folder.

The experimental results are shown in Table 2. The second and third columns denote the number of qubits in each protocol and the number of states in the reachable state space of each protocol under model checking, respectively. Notice that the number of states of each protocol under model checking is the same for both pure states and mixed states. The fourth and fifth columns denote the number of rewriting steps performed for each protocol and the

verification time when pure states are used to represent quantum states; and similarly for the last two columns when mixed states are used. Note that, in these experiments, the mixed state in each protocol consists of only one pure state with a certain probability in an ensemble.

Although all model checking experiments were completed in just a few moments, the number of rewriting steps and the verification time for mixed states is considerably larger than that for pure states. This is not surprising because we intentionally used an ensemble that contains only one pure state with a certain probability in these experiments, and the pure state is represented by a vector $|\psi\rangle$, while the mixed state is represented by a density operator $|\psi\rangle\langle\psi|$, which is a matrix. The calculation for a matrix is more expensive than that for a vector. However, with mixed states, we can present a statistical mixture of multiple pure states and eliminate the global phase compared with the pure states. Our symbolic model checking quantum circuits in Maude can handle both pure states and mixed states, showing its usefulness in quantum circuit verification.

6. Related Work

Gay et al. [14] introduced a method for employing classical model checkers, such as PRISM, a probabilistic model checker, to verify quantum protocols. In their approach, each quantum state is assigned a distinct number of identifiers and transitions from one unique number to another representing the operations of quantum gates and measurements. However, this method necessitates the prior enumeration of states and the computation of state transitions, which are subsequently encoded into a PRISM specification. Despite their development of a tool called PRISMGEN to automate this process, its practicality is limited in real-world scenarios, supporting only two or three qubits due to the exponential proliferation of state numbers. In contrast, our approach does not require the pre-enumeration of states, as quantum states are directly formalized using Dirac notation with scalar values. Furthermore, we utilize rewrite rules to represent the effects of quantum gates and measurements, rendering our approach capable of handling a larger number of qubits. For instance, we have successfully verified the correctness of Quantum Secret Sharing and Entanglement Swapping involving four qubits using our approach.

Our approach to symbolic quantum circuit model checking draws inspiration from a prior work by Yuan et al. [15], which closely resembles our methodology. However, it's worth noting that their approach primarily centers around theorem proving rather than model checking. They adopt Dirac notation and a set of rules to formalize quantum states, gates, measurements, and reason about quantum circuits within the Coq interactive theorem prover. Nevertheless, a notable distinction is that their approach often necessitates users to supply essential lemmas to facilitate the completion of their proofs, a task that is generally considered challenging. In contrast, our approach operates entirely autonomously, requiring no human intervention.

7. Conclusion

We have extended our symbolic approach to deal with mixed states using density operators and have developed a support tool in Maude. Several quantum communication protocols have been

successfully analyzed using our approach/support tool, including Superdense Coding, Quantum Teleportation, Quantum Secret Sharing, and Entanglement Swapping. This demonstrates the usefulness of our symbolic model checking quantum circuits with density operators in Maude. As one piece of future work, we would conduct more case studies, where a statistical mixture of multiple pure states is realistically presented in them.

Acknowledgments

The research was supported by JAIST Research Grant for Fundamental Research, by JST SICORP Grant Number JPMJSC20C2, and by JSPS KAKENHI Grant Number JP23K19959.

References

[1] P. Shor, Algorithms for quantum computation: discrete logarithms and factoring, in: Proceedings 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 124–134. doi:10.1109/SFCS.1994.365700.

[2] C. M. Do, K. Ogata, Symbolic model checking quantum circuits in maude, in: The 35th International Conference on Software Engineering and Knowledge Engineering, SEKE 2023, 2023, pp. 103–108. doi:10.18293/SEKE2023-014.

[3] P. A. M. Dirac, A new notation for quantum mechanics, Mathematical Proceedings of the Cambridge Philosophical Society 35 (1939) 416–418. doi:10.1017/S0305004100021162.

[4] M. Clavel, et al., All About Maude, volume 4350 of *Lecture Notes in Computer Science*, Springer, 2007. doi:10.1007/978-3-540-71999-1.

[5] J. Meseguer, Twenty years of rewriting logic, The Journal of Logic and Algebraic Programming 81 (2012) 721–781. doi:10.1016/j.jlap.2012.06.003.

[6] C. H. Bennett, S. J. Wiesner, Communication via one- and two-particle operators on einstein-podolsky-rosen states, Phys. Rev. Lett. 69 (1992) 2881–2884. doi:10.1103/PhysRevLett.69.2881.

[7] C. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, W. Wootters, Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels, Physical review letters 70 (1993) 1895–1899. doi:10.1103/PhysRevLett.70.1895.

[8] M. Hillery, V. Bužek, A. Berthiaume, Quantum secret sharing, Physical Review A 59 (1999) 1829–1834. doi:10.1103/physreva.59.1829.

[9] M. Żukowski, A. Zeilinger, M. A. Horne, A. K. Ekert, "Event-ready-detectors" Bell experiment via entanglement swapping, Phys. Rev. Lett. 71 (1993) 4287–4290. doi:10.1103/PhysRevLett.71.4287.

[10] M. A. Nielsen, I. L. Chuang, Quantum Computation and Quantum Information: 10th Anniversary Edition, Cambridge University Press, 2010. doi:10.1017/CBO9780511976667.

[11] T. Takagi, C. M. Do, K. Ogata, Automated quantum program verification in dynamic quantum logic (to appear), in: DaLí: Dynamic Logic – New trends and applications, 2023.

[12] E. M. Clarke, T. A. Henzinger, H. Veith, R. Bloem (Eds.), Handbook of Model Checking, Springer, 2018. doi:10.1007/978-3-319-10575-8.

[13] W. K. Wootters, W. H. Zurek, A single quantum cannot be cloned, Nature 299 (1982) 802–803. doi:10.1038/299802a0.
[14] S. Gay, R. Nagarajan, N. Papanikolaou, Probabilistic model–checking of quantum protocols, 2005. doi:10.48550/arXiv.quant-ph/0504007.
[15] W. Shi, Q. Cao, Y. Deng, H. Jiang, Y. Feng, Symbolic reasoning about quantum circuits in coq, J. Comput. Sci. Technol. 36 (2021) 1291–1306. doi:10.1007/s11390-021-1637-9.

Reachability Analysis of the Equivalence of Two Terms in Free Orthomodular Lattices

Tsubasa Takagi[1], Canh Minh Do[2] and Kazuhiro Ogata[2]

[1]*Tokyo Institute of Technology, Tokyo, Japan*
[2]*Japan Advanced Institute of Science and Technology, Nomi, Japan*

Abstract

Since 1936, the algebraic structure of quantum mechanics called orthomodular lattices have attracted many logicians' attention. Over its long history, various complex theorems have been manually proven. Some tools have been developed to automatically verify these theorems by means of checking the equivalence of two terms (the word problem). However, existing tools cannot deal with terms that consist of three or more free variables. Although the number of normal forms in free orthomodular lattices with three or more generators is infinite, that of two generators is only 96. To overcome this limitation, we transform the word problem into a reachability problem. Using this idea, we implement a support tool in Maude. It consists of a formal specification of free orthomodular lattices and an implementation of (1) the 96 normal forms of two generators and (2) a theorem describing when the distributive laws can be applied. The reachability analysis is performed using the `search` command in Maude to deal with terms that consist of even three or more variables. We conduct some case studies with the support tool to verify various complex theorems that existing tools cannot verify.

Keywords

Orthomodular Lattice, Word Problem, Reachability Analysis, Maude

1. Introduction

The algebraic structure of quantum mechanics has been discussed by many logicians since 1936 [1]. Among various algebras, orthomodular lattices (sometimes called quantum logic) [2] are particularly significant because the set of all closed subspaces of a Hilbert space forms an orthomodular lattice, which is called a Hilbert lattice [3].

So far, some theorems such as the Foulis-Holland theorem [4, 5] and a theorem [2, Corollary VII.7.7] about skew join and skew meet are verified [6] with the aid of computer programs [7, 8]. Furthermore, several new theorems regarding the verification of the equivalence of two terms have been obtained with the aid of these programs. For example, all associative operators [9], monotonic operators [10], and weak associative operators [11] in orthomodular lattices have been enumerated with their results, which could be hardly obtained without computer support.

The word problem for free orthomodular lattices remains an open problem in general [12]. To make matters worse, even if the word problem is solvable, it may not be implemented by finite convergent AC (associative and commutative) term rewriting systems. For example, it is

International Workshop on Formal Analysis and Verification of Post-Quantum Cryptographic Protocols 2023
takagi.t.ah@m.titech.ac.jp (T. Takagi); canhdo@jaist.ac.jp (C. M. Do); ogata@jaist.ac.jp (K. Ogata)
0000-0001-9890-1015 (T. Takagi); 0000-0002-1601-4584 (C. M. Do); 0000-0002-4441-3259 (K. Ogata)

known that there is no finite convergent AC term rewriting system for the equational theory of lattices [13]. However, although the word problem may not be solvable, it is still worth implementing a tool that can solve the word problem, not for any terms but for some specific terms of free orthomodular lattices, which consist of three or more free variables.

There are two existing programs [14, 7] for checking the equivalence of two terms in free orthomodular lattices. However, these programs cannot deal with terms that consist of three or more free variables. This is because there are infinite normal forms in three or more free generators, even though there are only 96 normal forms in the case of two free generators (Theorem 3.3). This limitation to some two free generators is fatal when proving theorems expressed by three or more generators in orthomodular lattices.

In this paper, we overcome this limitation by incorporating the idea of reachability analysis into a neither confluent nor terminating term rewriting system for free orthomodular lattices. The word problem is transformed into a reachability problem in the term rewriting system by searching for the reachable state space from an initial state. It is done by the breadth-first search (BFS) to find a solution in which the two terms in the word problem are rewritten into the same term after a finite number of rewrite steps, where each rewrite step can be regarded as a state transition. The reachability problem is conducted through a breadth-first search in an incremental way, which does not strictly require the reachable state space to be finite. However, the reachability analysis may not terminate in general.

Based on this idea, we implement a support tool in Maude, a rewriting logic-based specification/programming language that can deal with terms that consist of three or more free variables. The reachability analysis is performed using the `search` command, a reachability analyzer, in Maude. The support tool consists of a formal specification of free orthomodular lattices and an implementation of (1) the 96 normal forms of two generators (see Theorem 3.3) and (2) a theorem (see Theorem 3.4) describing when the distributive laws can be applied to check the word problem for free orthomodular lattices. Theorem 3.3 has been used in previous work [14, 7], while Theorem 3.4 is a new theorem proposed in this paper. To demonstrate the effectiveness of our approach, we verify the validity of some axioms with three free variables in several implication algebras [15, 16, 17, 18].

The rest of the paper is organized as follows. Section 2 reviews some kinds of lattices. Section 3 describes the theoretical background for the word problem for various free lattices and two significant theorems in orthomodular lattices. Section 5 presents the implementation of our tool in Maude. Section 6 shows some case studies. Section 7 compares our tool and existing tools. Finally, Section 8 concludes the paper together with some future directions.

2. Preliminaries

This section presents some kinds of lattices, such as distributive lattices, modular lattices, ortholattices, Boolean lattices, modular ortholattices, and orthomodular lattices. All of these lattices are defined using only equations. This makes it possible to implement them in algebraic specification languages (e.g., Maude [19]).

2.1. Distributive Lattice and Modular Lattice

Definition 2.1. A lattice is a triple $(L, \wedge, \vee)$ that consists of a non-empty set L and functions $\wedge : L \times L \to L$ and $\vee : L \times L \to L$ satisfying

1. (Associativity) $p \wedge (q \wedge r) = (p \wedge q) \wedge r$ and $p \vee (q \vee r) = (p \vee q) \vee r$,
2. (Commutativity) $p \wedge q = q \wedge p$ and $p \vee q = q \vee p$,
3. (Idempotency) $p \wedge p = p$ and $p \vee p = p$,
4. (Absorption) $p \wedge (p \vee q) = p$ and $p \vee (p \wedge q) = p$.

Given a lattice $(L, \wedge, \vee)$, a partial order (a binary relation that is reflexive, transitive, and antisymmetric) $\leq$ on L is defined by

$$\leq = \{(p, q) : p \wedge q = p\} = \{(p, q) : p \vee q = q\}.$$

Then, $p \wedge q$ is identical to the infimum (the greatest lower bound) of $\{p, q\}$, and $p \vee q$ is identical to the supremum (the least upper bound) of $\{p, q\}$.

The following properties of lattices are of particular significance.

Definition 2.2. A lattice $(L, \wedge, \vee)$ is said to be

- bounded if it has the least element (denoted by $\curlywedge$) and the greatest element (denoted by $\curlyvee$) under the partial order $\leq$.
- distributive if the distributive law holds:

  $$p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$$

  for any $p, q, r \in L$.
- modular if the modular law holds:

  $$p \wedge (q \vee (p \wedge r)) = (p \wedge q) \vee (p \wedge r)$$

  for any $p, q, r \in L$.

Example 2.3 (Powerset Lattice)**.** Let $\wp(X)$ be the powerset of a set X. Then, $(\wp(X), \cap, \cup)$ called the powerset lattice on X is a distributive lattice.

Example 2.4. Let $\mathcal{G} = (G, +)$ be an additive group, and $L(\mathcal{G})$ be the set of all subgroups of $\mathcal{G}$. Then, $(L(\mathcal{G}), \cap, +)$ is a modular lattice, where $+$ on $L(\mathcal{G})$ is defined by

$$G_1 + G_2 = \{g_1 + g_2 : g_1 \in G_1, g_2 \in G_2\}.$$

Note that $G_1 + G_2$ is the smallest subgroup of $\mathcal{G}$ containing $G_1 \cup G_2$.

By using the partial order $\leq$, the modular law can be rephrased as follows [20]:

$$p \leq r \text{ implies } p \vee (q \wedge r) = (p \vee q) \wedge r.$$

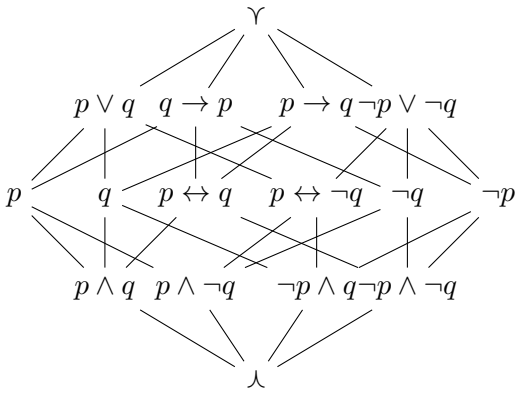
2.2. Ortholattice and Orthomodular Lattice

Definition 2.5. An ortholattice (also called an orthocomplemented lattice) is a bounded lattice equipped with an orthocomplementation. An orthocomplementation on a bounded lattice $(L, \wedge, \vee)$ is a function $\neg : L \to L$ such that

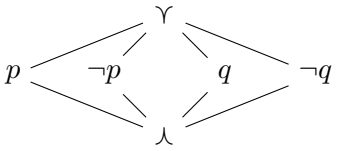
1. $p \wedge \neg p = \curlywedge$ and $p \vee \neg p = \curlyvee$,
2. $\neg\neg p = p$,
3. $\neg(p \wedge q) = \neg p \vee \neg q$ and $\neg(p \vee q) = \neg p \wedge \neg q$,

for any $p, q \in L$. In particular, distributive ortholattices are called Boolean lattices.

Example 2.6 (Four-dimensional Hypercube)**.** The lattice depicted in the following Hasse diagram is a Boolean lattice, where $p \to q$ and $p \leftrightarrow q$ are abbreviations for $\neg p \vee q$ and $(p \wedge q) \vee (\neg p \wedge \neg q)$, respectively. This lattice is called the four-dimensional hypercube and is denoted by $\mathbf{2}^4$.



Example 2.7 (Chinese Lantern)**.** The lattice depicted in the following Hasse diagram is a (non-distributive) modular ortholattice. This lattice is called the Chinese lantern and is denoted by MO_2.



Example 2.8 (Boolean Powerset Lattice)**.** Let ^{c} be the set complementation in a set X. Then, $(\wp(X), \subseteq, ^c)$ is a Boolean lattice and is called the Boolean powerset lattice on X.

Example 2.9 ([3, Proposition 4.3])**.** Let $\mathcal{H} = (H^{\text{fin}}, +, \cdot)$ be a finite-dimensional Hilbert space over the complex numbers and $\mathcal{C}(\mathcal{H})$ be the set of all (closed) subspaces of $\mathcal{H}$. Then, $(\mathcal{C}(\mathcal{H}), \cap, +, ^\perp)$ is a modular ortholattice, where $+$ on $\mathcal{H}$ is defined by

$$V_1 + V_2 = \{v_1 + v_2 : v_1 \in V_1, v_2 \in V_2\},$$

and $v' \in V^\perp$ if and only if the inner product of v' and v is 0 for any $v \in V$.

Note that $\mathcal{C}(\mathcal{H})$ is not modular if the dimension of $\mathcal{H}$ is infinite [3, Proposition 4.4]. However, quantum mechanics are formulated by an infinite dimensional Hilbert space in general. This motivates us to weaken the modular law to the orthomodular law below.

Definition 2.10. An orthomodular lattice is an ortholattice $(L, \wedge, \vee, \neg)$ satisfying the orthomodular law [3, Proposition 3.5]:

$$p \leq q \text{ implies } q = p \vee (\neg p \wedge q),$$

for any $p, q \in L$

For any Hilbert space $\mathcal{H}$, regardless of its dimension, $(\mathcal{C}(\mathcal{H}), \cap, +, ^{\perp})$ is an orthomodular lattice [3, Proposition 4.5]. Thus, modular ortholattices and orthomodular lattices are different.

By the following theorem, the class of orthomodular lattices is characterized only by equations.

Theorem 2.11. *Let $(L, \wedge, \vee, \neg)$ be an ortholattice. $(L, \wedge, \vee, \neg)$ is an orthomodular lattice if and only if*

$$p \vee q = p \vee (\neg p \wedge (p \vee q)),$$

for any $p, q \in L$.

Proof. See [2, Theorem II.5.1], and use the absorption law. □

Definition 2.12. Let $\mathcal{L}_1 = (L_1, \wedge_1, \vee_1, \neg_1)$ and $\mathcal{L}_2 = (L_2, \wedge_2, \vee_2, \neg_2)$ be ortholatices. Then,

$$\mathcal{L}_1 \times \mathcal{L}_2 = (L_1 \times L_2, \wedge, \vee, \neg)$$

defined by

$$\begin{aligned}(p_1, p_2) \wedge (q_1, q_2) &= (p_1 \wedge_1 q_1, q_1 \wedge_2 q_2),\\ (p_1, p_2) \vee (q_1, q_2) &= (p_1 \vee_1 q_1, q_1 \vee_2 q_2),\\ \neg(p_1, p_2) &= (\neg_1 p_1, \neg_2 p_2)\end{aligned}$$

is an ortholattice and is called the direct product of $\mathcal{L}_1$ and $\mathcal{L}_2$.

3. Theoretical Background

The main contribution of this paper is to implement a support tool to verify the equivalence of two terms in free orthomodular lattices. Its principle is based on two fundamental theorems explained in this section. Although the first theorem (Theorem 3.3) has been employed for the implementation in the previous work [14, 7], the second theorem (Theorem 3.4) is a new theorem to be proved in this paper and has not been used in previous work. The second theorem allows verifying the equivalence between a broader class of terms than existing work.

3.1. Free Algebra

Definition 3.1. Let $\mathcal{C}$ be a non-empty class of algebras. An algebra $F_X \in \mathcal{C}$ is called a free algebra in $\mathcal{C}$ generated by X, if F_X is generated by $X \subseteq F_X$ and every function $V : X \to A \in \mathcal{C}$ can be uniquely extended to a homomorphism $\widehat{V} : F_X \to A$.

Example 3.2. The Boolean lattice $\mathbf{2}^4$ in Example 2.6 is a free Boolean lattice with two generators p and q.

In general, the free Boolean lattice with n generators is the Boolean lattice $\mathbf{2}^{2^n}$ that consists of 2^{2^n} elements [20, Theorem Ⅲ.7].

3.2. The Word Problem for Various Free Lattices

The problem of deciding whether or not two given terms (words) of algebras are equivalent is called the word problem for the algebras. One possible way to check it is to try to reduce the two given terms to the same term in a finite number of steps.

The word problem for various kinds of lattices is one of the central topics in lattice theory. It has been shown that the word problem for free lattices is solvable [21] (for more details, see [22]). The following results have been obtained for various free lattices with some restrictions.

- The word problem for free distributive lattices is solvable [23].
- The word problem for free modular lattices with $n \leq 3$ generators is solvable (because the free modular lattice with three generators has only 28 terms [24]), and that for free modular lattices with $n \geq 4$ generators is unsolvable [25].
- The word problem for free ortholattices is solvable [26].
- The word problem for the free modular ortholattice with $n \leq 2$ generators is solvable (because the free modular ortholattice with two generators has only 96 terms [27]), and that for general free modular ortholattices remains an open problem [28].
- The word problem for the free orthomodular lattice with $n \leq 2$ generators is solvable (because the free orthomodular lattice with two generators has only 96 terms [2, Theorem Ⅲ.2.8]), and that for general free modular ortholattices remains an open problem [12].

From the perspective of quantum mechanics, the word problem for free orthomodular lattices is the most significant among the above-listed word problems. This is because the set of all closed subspaces (experimental propositions [1]) of a Hilbert space is an orthomodular lattice [3, Proposition 4.5].

3.3. Two Fundamental Theorems for Orthomodular Lattices

To solve the word problem for some specific free orthomodular lattices, we use two fundamental theorems for orthomodular lattices.

The first fundamental theorem states that there are only 96 normal forms in the free orthomodular lattice with two generators. This theorem has been used in previous work [14, 7].

Theorem 3.3. *$MO_2 \times \mathbf{2}^4$ is isomorphic to the free orthomodular lattice with two generators.*

Proof. See [2, Theorem III.2.8]. □

Because the number of normal forms in MO_2 is 6 (recall Example 2.7) and that in $\mathbf{2}^4$ is 16 (recall Example 2.6), that of $MO_2 \times \mathbf{2}^4$ is $6 \times 16 = 96$.

Also, Theorem 3.3 clarifies the relationship between any two normal forms. Let a be an element of MO_2 distinct from $\curlywedge$ and $\curlyvee$, and $\mathbf{2} = (\{\curlywedge, \curlyvee\}, \wedge, \vee, \neg)$ be the two-element Boolean lattice. Then, for example, p (referred to as $\mathfrak{p}_{22}$) and $\neg p \wedge (p \vee q)$ (referred to as $\mathfrak{p}_{68}$) are normal forms corresponding to $(a, \curlyvee, \curlyvee, \curlywedge, \curlywedge)$ and $(\neg a, \curlywedge, \curlywedge, \curlyvee, \curlywedge)$ in $MO_2 \times \mathbf{2} \times \mathbf{2} \times \mathbf{2} \times \mathbf{2}$, which is isomorphic to $MO_2 \times \mathbf{2}^4$, respectively [2, Fig. 18]. Note that a be an element of MO_2. By definitions of MO_2 and $\mathbf{2}$,

$$(a, \curlyvee, \curlyvee, \curlywedge, \curlywedge) \vee (\neg a, \curlywedge, \curlywedge, \curlyvee, \curlywedge) = (\curlyvee, \curlyvee, \curlyvee, \curlyvee, \curlywedge).$$

Here, $(\curlyvee, \curlyvee, \curlyvee, \curlyvee, \curlywedge)$ corresponds to $p \vee q$ [2, Fig. 18] (referred to as $\mathfrak{p}_{92} = \mathfrak{p}_{22} \vee \mathfrak{p}_{68}$), which is also one of the 96 normal forms. Therefore, we obtain the result that

$$p \vee (\neg p \wedge (p \vee q)) = p \vee q$$

holds in orthomodular lattices. In fact, this equation is the orthomodular law (recall Theorem 2.11). A list describing the relationship between any two normal forms is in [2, Fig. 18].

The second fundamental theorem tells us when the distributive law can be applied in orthomodular lattice. This is a new theorem proposed in this paper.

Theorem 3.4. *Let $(L, \wedge, \vee, \neg)$ be an orthomodular lattice. Then, the following are equivalent:*

1. $p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$;
2. $p \wedge (\neg p \vee q) = p \wedge q$.

Dually, the following are also equivalent:

1. $p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$;
2. $p \vee (\neg p \wedge q) = p \vee q$.

Proof. We only prove the first part of the theorem. The dual part is shown similarly.

Proof of (1) ⇒ (2). By (1),

$$p \wedge (\neg p \vee q) = (p \wedge \neg p) \vee (p \wedge q) = p \wedge q.$$

Proof of (2) ⇒ (1). Recall that the distributive inequality

$$(p \wedge q) \vee (p \wedge r) \leq p \wedge (r \vee q)$$

always holds even in non-distributive lattices (see [29, Exercise 2.9 (ii)]). Thus, $p \wedge (q \vee r)$ is an upper bound of $\{p \wedge q, p \wedge r\}$. Thus, it remains to show that $p \wedge (q \vee r)$ is the least upper bound of $\{p \wedge q, p \wedge r\}$: $p \wedge (q \vee r) \leq s$ for any upper bound s of $\{p \wedge q, p \wedge r\}$. Clearly,

$$(p \wedge (q \vee r)) \wedge s \leq p \wedge (q \vee r).$$

By applying the orthomodular law (Definition 2.10),

$$p \wedge (q \vee r) = ((p \wedge (q \vee r)) \wedge s) \vee ((\neg((p \wedge (q \vee r)) \wedge s)) \wedge (p \wedge (q \vee r))).$$

Hence, it suffices to show

$$(\neg((p \wedge (q \vee r)) \wedge s)) \wedge (p \wedge (q \vee r)) = \curlywedge.$$

Because s is an upper bound of $\{p \wedge q, p \wedge r\}$, we obtain $p \wedge q \leq s$ and $p \wedge r \leq s$. Therefore,

$$p \wedge q = (p \wedge q) \wedge s \leq (p \wedge (q \vee r)) \wedge s$$

and

$$p \wedge r = (p \wedge r) \wedge s \leq (p \wedge (q \vee r)) \wedge s.$$

That is,

$$\neg((p \wedge (q \vee r)) \wedge s) \leq \neg(p \wedge q) = \neg p \vee \neg q$$

and

$$\neg((p \wedge (q \vee r)) \wedge s) \leq \neg(p \wedge r) = \neg p \vee \neg r.$$

It follows from the assumption (2) that

$$\begin{aligned}&\neg((p \wedge (q \vee r)) \wedge s) \wedge (p \wedge (q \vee r)) \leq (\neg p \vee \neg q) \wedge (p \wedge (q \vee r)) \\ &= ((\neg p \vee \neg q) \wedge p) \wedge (q \vee r) = (p \wedge \neg q) \wedge (q \vee r).\end{aligned}$$

Similarly, we have

$$\neg((p \wedge (q \vee r)) \wedge s) \wedge (p \wedge (q \vee r)) \leq (p \wedge \neg r) \wedge (q \vee r).$$

Consequently,

$$\begin{aligned}&\neg((p \wedge (q \vee r)) \wedge s) \wedge (p \wedge (q \vee r)) \leq ((p \wedge \neg q) \wedge (q \vee r)) \wedge ((p \wedge \neg r) \wedge (q \vee r)) \\ &= (p \wedge (\neg q \wedge \neg r)) \wedge (q \vee r) = p \wedge (\neg(q \vee r) \wedge (q \vee r)) = p \wedge \curlywedge = \curlywedge.\end{aligned}$$

□

In the next section, we combine these two fundamental theorems to implement a support tool.

4. Implementation

We develop a support tool in Maude [19] to check the word problem for free orthomodular lattices. Maude is a high-performance specification/programming based on rewriting logic [30]. The language can directly specify order-sorted equational logic and rewriting logic, and it provides several formal analysis methods, such as reachability analysis through a breadth-first search of the reachable state space from a given state. The reachability analysis is performed using the `search` command in Maude. In the support tool, we formally specify free orthomodular lattices, and develop an implementation of Theorem 3.3 and Theorem 3.4 to check the word problem using the `search` command. This section gives the syntax of the Maude language in a nutshell (see [19] for more details) and briefly describes how the tool is developed.

4.1. Overview of Maude

Functional modules

A functional module $\mathcal{M}$ specifies an order-sorted equational logic theory (Σ, E) with the syntax: **fmod** $\mathcal{M}$ **is** (Σ, E) **endfm**, where Σ is an order-sorted signature and E is the collection of equations in the functional module. (Σ, E) may contain a set of declarations as follows:

- importations of previously defined modules (**protecting** ... or **extending** ... or **including** ...)
- declarations of sorts (**sort** s **.** or **sorts** $s\ s'$ **.**)
- subsort declarations (**subsort** $s < s'$ **.**)
- declarations of function symbols (**op** f **:** $s_1\ \ldots\ s_n \rightarrow s$ $[att_1 \ldots att_k]$ **.**)
- declarations of variables (**vars** $v\ v'$ **.**)
- declarations of unconditional equations (**eq** $t = t'$ **.**)
- declarations of conditional equations (**ceq** $t = t'$ **if** $cond$ **.**)

where $s, s_1, \ldots, s_n$ are sort names, v, v' are variable names, t, t' are terms, $cond$ is a conjunction of equations (i.e., $t = t'$ and/or t => t'), and $att_1, \ldots att_k$ are equational attributes. Equations are used as *equational rules* to perform the simplification in which instances of the left-hand side pattern are subterms of a subject term replaced by the corresponding instances of the right-hand side. The process is called *term rewriting*, and the result of simplifying a term is called its *normal form*.

System modules

A system module $\mathcal{R}$ specifies a rewrite theory (Σ, E, R) with the syntax: **mod** $\mathcal{R}$ **is** (Σ, E, R) **endm**, where Σ and E are the same as those in an equational theory, and R is the collection of rewrite rules in the system module. (Σ, E, R) may contain all possible declarations in (Σ, E) and rewrite rules in R as follows:

- declarations of unconditional rewrite rules (**rl** $[lbl]$ **:** u => v **.**)
- declarations of conditional rewrite rules (**crl** $[lbl]$ **:** u => v **if** $cond$ **.**)

where lbl is the name of a rewrite rule, u, v are terms, and $cond$ is a conjunction of equations and/or rewrites (e.g., t => t'). rewrite rules are executed as rewriting from left to right modulo the equations in the system module and are regarded as *local transition rules*, making possible many state transitions from a given state as concurrent changes of a system.

Reachability analysis

The word problem of two terms t_1 and t_2 can be transformed into a reachability problem. Suppose that t `=?=` t `=` `true` for any term t, then the word problem is to check whether `true` is reachable from t_1 `=?=` t_2 referred to as t_1 `=?=` t_2 $\rightarrow^*_{R,E}$ `true`, where R, E are the rewrite rules and equations in the specification of a system concerned. If that is the case, the two terms t_1 and t_2 are reduced to the same term in a finite number of rewrite steps, meaning that the word problem is solved for t_1 and t_2.

For the sake of simplicity, we use a Boolean lattice $\mathcal{L} = (L, \wedge, \vee, \neg)$ as an example to demonstrate how to use the `search` command in Maude to tackle the reachability problem to solve the word problem for $\mathcal{L}$. The following system module formally specifies $\mathcal{L}$.

```
mod BOOLEAN-LATTICE is
  sort Lattice .
  ops bot top : -> Lattice [ctor] .
  op not_ : Lattice -> Lattice [prec 31] .
  op _and_ : Lattice Lattice -> Lattice [assoc comm prec 32] .
  op _or_ : Lattice Lattice -> Lattice [assoc comm prec 33] .
  op _|->_ : Lattice Lattice -> Lattice [prec 34] .
  vars P Q R : Lattice .
  rl [imply] : P |-> Q => not P or Q .
  --- Idempotent Laws
  eq P and P = P .    eq P or P = P .
  --- Absorption Laws
  eq P and (P or Q) = P .    eq P or (P and Q) = P .
  --- Orthocomplementation
  eq P and not P = bot .    eq P or not P = top .
  eq not(not P) = P .
  eq not (P and Q) = not P or not Q .
  eq not (P or Q) = not P and not Q .
  --- Distributive Laws
  eq P or (Q and R) = (P or Q) and (P or R) .
  eq P and (Q or R) = (P and Q) or (P and R) .
  --- Additional Identities
  eq not top = bot .    eq not bot = top .
  eq P and top = P .    eq P and bot = bot .
  eq P or bot = P .     eq P or top = top .
  --- Checking Equivalence of Two Lattices
  op true : -> Lattice [ctor] .
  op _=?=_ : Lattice Lattice -> Lattice .
  eq P =?= P = true .
endm
```

where the `_and_`, `_or_`, `not_`, `bot`, `top` operators represent $\wedge, \vee, \neg, \curlywedge, \curlyvee$ in the Boolean lattice $\mathcal{L}$, respectively. `P`, `Q`, and `R` are Maude variables of the sort `Lattice` representing elements of L. The `assoc` and `comm` attributes denote associative and commutative properties. The `prec_` attribute, where `_` is a natural number, denotes the precedences of operators for parsing in Maude (e.g., `not P and Q or R` is parsed as `((not P)and Q)or R` without using parentheses). The idempotent laws, absorption laws, distributive laws, involution, complements,

identities, and De Morgan's laws are defined in the form of equations, while the implication is defined in the form of a rewrite rule (see the rule `imply`). Although we should define the implication using an equation, we intentionally use the rewrite rule `imply` intending to make concurrent changes in this example. For example, if a term consists of at least two sub-terms of in the form of implication, either one of the two sub-terms can be rewritten using the rewrite rule `imply`, making concurrent changes for the subject term. This is because rewrite rules can be used instead of equations for equational reasoning. The `_=?=_` operator is used for constructing the initial term of the word problem in reachability analysis such that whenever two terms at both sides of the operator are equivalent, it becomes `true`, which is a special element of P denoting the validity of the word problem.

We define some arbitrary elements of P in the form of some operators (or fresh constants) in the following module:

```
mod BOOLEAN-LATTICE-TEST is
  pr BOOLEAN-LATTICE .
  ops p q r : -> Lattice .
endm
```

We can verify the axiom (I3) of Abbott's implication algebra [15] using the following command in Maude.

```
search [1] p |-> (q |-> r) =?= q |-> (p |-> r) =>* true .
```

The command says that we would like to search on the reachable state space of $\mathcal{L}$ from the initial state `p |-> (q |-> r)=?= q |-> (p |-> r)` to check whether there is one solution in which `true` is reachable from the initial state after zero or more rewrite steps. Note that given the initial state as a term, the rewrite rule `imply` can be applied at four positions of the term, making four possible successor states from the initial state as follows:

```
(1) not p or (q |-> r) =?= q |-> p |-> r
(2) p |-> (not q or r) =?= q |-> p |-> r
(3) p |-> q |-> r =?= not q or (p |-> r)
(4) p |-> q |-> r =?= q |-> (not p or r)
```

In this way, the reachable state space of $\mathcal{L}$ from the initial state is constructed. The result of the command returns a solution and then we can conclude that the axiom (I3) is verified using the `search` command in Maude.

5. A Support Tool

We develop a support tool in Maude and use the `search` command to check the word problem for free orthomodular lattices. The tool is publicly available at https://github.com/canhminhdo/FOM. The basic idea of developing the tool is similar to what we have described above on how to use the `search` command to verify the axiom (I3) in Maude. Moreover, we develop an implementation of Theorem 3.3 and Theorem 3.4 in Maude to check the word problem for free orthomodular lattices.

First, we prepare a formal specification of free orthomodular lattices in Maude, where some properties are specified, such as associativity, commutativity, idempotency of $\wedge$ and $\vee$, the

absorption laws, De Morgan's laws, and so on. All properties are described by means of equations in Maude. Let E_{FOM} be the sets of equations from specifying free orthomodular lattices.

Second, we formalize Theorem 3.4 with the compatibility relation using rewrite rules. If the subject term can be applied by the rewrite rules of Theorem 3.4 at multiple positions, the order of applying the rewrite rules may produce different results in the end. Therefore, we specify Theorem 3.4 using rewrite rules instead of equations to make concurrent changes in our system so that all possible cases can be taken into account using reachability analysis. Moreover, we may need to apply distributive laws from not only left to right but also right to left. If we use equations to do so, it will make an infinite computation because of a loop caused by constantly applying the equations from left to right and vice versa for a subject term if distributive laws are applicable. Therefore, we use rewrite rules to specify Theorem 3.4. Let R_{DIST} be the set of rewrite rules obtained from specifying Theorem 3.4.

Third, we formalize the 96 normal forms (Theorem 3.3), and then automatically generate all equations for each $\mathfrak{p}_i \wedge \mathfrak{p}_j$, $\mathfrak{p}_i \vee \mathfrak{p}_j$, and $\neg\mathfrak{p}_i$ that is again an element $\mathfrak{p}_k$, where $\mathfrak{p}_i, \mathfrak{p}_j, \mathfrak{p}_m$, and $\mathfrak{p}_k$ are elements of the 96 normal forms in [2, Fig. 18]. Because $\neg\mathfrak{p}_i$ can be obtained through applying De Morgan's laws specified in E_{FOM}, then we do not need to construct equations for $\neg\mathfrak{p}_i$ of 96 normal forms. Hence, we only focus on generating equations for $\mathfrak{p}_i \wedge \mathfrak{p}_j$ and $\mathfrak{p}_i \vee \mathfrak{p}_j$. For example, $\mathfrak{p}_{92} = \mathfrak{p}_{22} \vee \mathfrak{p}_{68}$ (see Subsection 3.3 for more details), and thus we can obtain the following equation:

```
eq P or (not P and (P or Q)) = P or Q .
```

where `P` and `Q` are variables representing elements of free orthomodular lattices. Let E_{NF} be the set of equations obtained from generating all equations of the 96 normal forms. Note that the left-hand side of each generated equation is simplified by using E_{FOM} beforehand and unnecessary equations (e.g., `eq P = P .`) are removed. Hence, there are 5,520 distinct equations in E_{NF} generated automatically by the support tool.

Now the word problem is rephrased as the reachability problem t_1 `=?=` t_2 $\rightarrow^*_{R,E}$ `true` with the relation $\rightarrow^*_{R,E}$ that is $\rightarrow^*_{R_{DIST}, E_{FOM} \cup E_{NF}}$.

6. Case Studies

We apply the support tool to show that some axioms with three free variables in several implication algebras [16, 18, 15, 17] are valid. In these experiments, we used one node in computing servers available at our institute that carries a 2.8 GHz microprocessor with 64 cores and 1.5 TB memory of RAM.

Before that, we briefly explain what implication algebras are. The operators $\wedge$, $\vee$, and $\neg$ in orthomodular lattices (also in Boolean lattices) correspond to conjunction, disjunction, and negation, respectively. In Boolean lattices $(L, \wedge, \vee, \neg)$, the operator $\rightarrow$ (material implication) on P is defined by $p \rightarrow q = \neg p \vee q$. This operator $\rightarrow$ satisfies all the minimal requirements for implication called the minimal implicative criteria:

1. $p \leq q$ implies $p \rightarrow q = \top$;
2. (Modus Ponens) $p \wedge (p \rightarrow q) \leq q$;

3. (Modus Tollens) $\neg q \wedge (p \to q) \leq \neg p$.

On the other hand, in orthomodular lattices, there are three implication operators $\rightsquigarrow$ (Sasaki implication [31] or quasi implication [15]), $\rightarrowtail$ (Dishkant implication [18] or ortho-implication [16]), and $\twoheadrightarrow$ (relevance implication [18]) that satisfy the minimal implicative criteria [3, Chapter 8] defined by

$$p \rightsquigarrow q = \neg p \vee (p \wedge q), \quad p \rightarrowtail q = (\neg p \wedge \neg q) \vee q,$$
$$p \twoheadrightarrow q = ((p \wedge q) \vee (\neg p \wedge q)) \vee (\neg p \wedge \neg q).$$

Note that all these three implication operators coincide with $p \to q$ in Boolean lattices (using the distributive law).

For each implication satisfying the minimal implicative criteria, algebras that the only operator is the implication (and $\curlywedge$) have been proposed and are called implication algebras. All the axioms of these implication algebras are satisfied in orthomodular lattices by translating an implication into a term in orthomodular lattices (for example, $p \rightsquigarrow q$ is translated into $\neg p \vee (p \wedge q)$).

Because any axioms that consist of two variables can be verified by existing tool [14, 7], we confine our attention to axioms that consist of three variables in several implication algebras:

- The axiom (Q2) of quasi-implication algebra [15]:

$$(p \rightsquigarrow q) \rightsquigarrow (p \rightsquigarrow r) = (q \rightsquigarrow p) \rightsquigarrow (q \rightsquigarrow r).$$

- The axiom (O2) of ortho-implication algebra [16]:

$$p \rightarrowtail ((q \rightarrowtail p) \rightarrowtail r) = p \rightarrowtail r.$$

- The axiom (O5) of orthomodular implication algebra [17]:

$$(((p \rightarrowtail q) \rightarrowtail q) \rightarrowtail r) \rightarrowtail (p \rightarrowtail r) = \curlyvee$$

- The axiom (O6) of orthomodular implication algebra [17]:

$$\begin{aligned}&(((((((((p \rightarrowtail q) \rightarrowtail q) \rightarrowtail r) \rightarrowtail r) \rightarrowtail r) \rightarrowtail p) \rightarrowtail p) \rightarrowtail r) \rightarrowtail p) \rightarrowtail p \\ &= (((p \rightarrowtail q) \rightarrowtail q) \rightarrowtail r) \rightarrowtail r\end{aligned}$$

- The axiom (J4) of Sasaki implication algebra [18]:

$$\begin{aligned}&p \rightsquigarrow ((p \rightsquigarrow ((q \rightsquigarrow ((q \rightsquigarrow r) \rightsquigarrow \curlywedge)) \rightsquigarrow \curlywedge)) \rightsquigarrow \curlywedge) \\ &= r \rightsquigarrow ((r \rightsquigarrow ((p \rightsquigarrow ((p \rightsquigarrow q) \rightsquigarrow \curlywedge)) \rightsquigarrow \curlywedge)) \rightsquigarrow \curlywedge).\end{aligned}$$

- The axiom (K5) of Dishkant implication algebra [18]:

$$(((p \rightarrowtail q) \rightarrowtail q) \rightarrowtail r) \rightarrowtail r = (p \rightarrowtail ((q \rightarrowtail r) \rightarrowtail r)) \rightarrowtail ((q \rightarrowtail r) \rightarrowtail r).$$

- The axiom (L6) of relevance implication algebra [18]:

$$(((p \twoheadrightarrow q) \twoheadrightarrow q) \twoheadrightarrow r) \twoheadrightarrow r = (p \twoheadrightarrow ((q \twoheadrightarrow r) \twoheadrightarrow r)) \twoheadrightarrow ((q \twoheadrightarrow r) \twoheadrightarrow r).$$

Table 1
Experimental results for validating some axioms with our support tool

Target Axiom	Time
The axiom (Q2) in [15]	1,213ms
The axiom (O2) in [16]	736ms
The axiom (O5) in [17]	705ms
The axiom (O6) in [17]	716ms
The axiom (J4) in [18]	715ms
The axiom (K5) in [18]	723ms
The axiom (L6) in [18]	6d:19h:40m

The experimental data for checking these axioms are shown in Table 1. The first and second columns denote the name of each axiom and the time taken to validate each axiom using our support tool. For the first six axioms, our tool can quickly verify their validity for a few moments, which is almost impossible to do so without the aid of computer programs, especially the axioms (O5), (J4), and (K5). For the last axiom (L6), our tool needs to spend six days, 19 hours, and 40 minutes to verify its validity because of its complexity, making a huge state space in the reachability analysis. In summary, we can conclude the validity of axioms (Q2), (O2), (O5), (J4), and (K5) using our support tool. Transforming the word problem into a reachability problem and the use of not only the 96 normal forms (Theorem 3.3) but also Theorem 3.4 for the applicability of distributive laws make our tool extremely useful and distinguishable from existing tools for checking the word problem in free orthomodular lattices.

7. Related Work

In [14], a program is implemented to reduce a term in the free orthomodular lattice $F(p, q)$ with two generators p and q to the normal (canonical) form. It relies on the fact that $F(p, q)$ consists of the 96 normal forms (Theorem 3.3) [2].

After that, in [7], a program that can reduce a term in the free orthomodular lattice $F(p, q, r_1, \ldots, r_n)$ generated by $n + 2$ terms $p, q, r_1, \ldots, r_n$, where $1 \leq n \leq 9$, and each r_i commutes with all other generators. Note that p is said to commute with q, denoted by pCq, if

$$p = (p \wedge q) \vee (p \wedge \neg q).$$

The implementation in [7] relies on the fact that $F(p, q, r_1, \ldots, r_n)$ is isomorphic to the direct product of 2^n copies of $F(p, q)$ [32]. Different from the program in [14], that in [7] makes it possible to verify a much broader class of theorems in orthomodular lattice theory [6], such as the Foulis-Holland theorem [4, 5] and a theorem [2, Corollary VII.7.7] about skew join and skew meet. In addition, some problems, such as verification of associative operators [9], monotonic operators [10], and weak associative operators [11] of operations on orthomodular lattices, which can be reduced to verification of terms in $F(p, q, r_1, \ldots, r_n)$ are solved using the program in [7].

Although both implementations in [14] and [7] can help to solve various problems, they still have limitations: that in [14] can only deal with terms consisting of two generators, and that

in [7] can only deal with terms in a free orthomodular lattice with restricted generators.

Our tool can overcome these limitations when three or more generators are allowed to be used in terms. Based on Theorem 3.3 and 3.4, our tool effectively checks the word problem in free orthomodular lattices without any restrictions and can cover a broader class of the word problems compared to existing tools.

8. Conclusion

Using a reachability analysis, we have described how to develop the support tool for checking the word problem with three or more generators for free orthomodular lattices. The tool implemented in Maude consists of the formal specification of free orthomodular lattices and the implementation of two fundamental theorems (Theorem 3.3 and Theorem 3.4). We have used the `search` command in Maude to conduct the reachability analysis to conclude the validity of some complicated theorems, demonstrating the power of our support tool.

Acknowledgments

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.

References

[1] G. Birkhoff, J. von Neumann, The logic of quantum mechanics, Annals of mathematics 57 (1936) 823–843.

[2] L. Beran, Orthomodular lattices: algebraic approach, volume 18 of *Mathematics and its Applications*, 1985.

[3] M. Rédei, Quantum logic in algebraic approach, volume 91 of *Fundamental Theories of Physics*, Springer, 1998.

[4] D. J. Foulis, A note on orthomodular lattices, Portugaliae mathematica 21 (1962) 65–72.

[5] S. S. Holland, A radon-nikodym theorem in dimension lattices, Transactions of the American Mathematical Society 108 (1963) 66–87.

[6] M. Hyčko, M. Navara, Decidability in orthomodular lattices, International Journal of Theoretical Physics 44 (2005) 2239–2248.

[7] M. Hyčko, Implications and equivalences in orthomodular lattices, Demonstratio Mathematica 38 (2005) 777–792.

[8] N. D. Megill, M. Pavičić, Quantum implication algebras, International Journal of Theoretical Physics 42 (2003) 2807–2822.

[9] J. Gabriëls, M. Navara, Associativity of operations on orthomodular lattices, Mathematica Slovaca 62 (2012) 1069–1078.

[10] J. J. Gabriëls, M. Navara, Computer proof of monotonicity of operations on orthomodular lattices, Information Sciences 236 (2013) 205–217.

[11] S. M. Gagola, J. J. Gabriëls, M. Navara, Weaker forms of associativity in orthomodular lattices, Algebra universalis 73 (2015) 249–266.
[12] G. Bruns, J. Harding, Algebraic aspects of orthomodular lattices, in: Current research in operational quantum logic, volume 111 of *Fundamental Theories of Physics*, 2000, pp. 37–65.
[13] R. Freese, J. Ježek, J. B. Nation, Term rewrite systems for lattice theory, Journal of Symbolic Computation 16 (1993) 279–288.
[14] N. D. Megill, M. Pavičić, Orthomodular lattices and a quantum algebra, International Journal of Theoretical Physics 40 (2001) 1387–1410.
[15] G. M. Hardegree, Quasi-implication algebras, part I: Elementary theory, Algebra Universalis 12 (1981) 30–47.
[16] J. C. Abbott, Orthoimplication algebras, Studia Logica 35 (1976) 173–177.
[17] I. Chajda, R. Halaš, H. Länger, Orthomodular implication algebras, International Journal of Theoretical Physics 40 (2001) 1875–1884.
[18] G. N. Georgacarakos, Equationally definable implication algebras for orthomodular lattices, Studia Logica 39 (1980) 5–18.
[19] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. L. Talcott (Eds.), All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic, volume 4350 of *Lecture Notes in Computer Science*, 2007.
[20] G. Birkhoff, Lattice theory, volume 25, third ed., American Mathematical Society, 1940.
[21] P. M. Whitman, Free lattices, Annals of Mathematics 42 (1941) 325–330.
[22] R. Freese, J. Ježek, J. B. Nation, Free lattices, volume 42 of *Mathematical Surveys and Monographs*, American Mathematical Society, 1995.
[23] K. Takeuchi, The word problem for free distributive lattices, Journal of the Mathematical Society of Japan 21 (1969) 330–333.
[24] R. Dedekind, Ueber die von drei moduln erzeugte dualgruppe, Mathematische Annalen 53 (1900) 371–403.
[25] C. Herrmann, On the word problem for the modular lattice with four free generators, Mathematische Annalen 265 (1983) 513–527.
[26] G. Bruns, Free ortholattices, Canadian Journal of Mathematics 28 (1976) 977–985.
[27] J. Kotas, An axiom system for the modular logic, Studia Logica 21 (1967) 17–38.
[28] M. S. Roddy, On the word problem for orthocomplemented modular lattices, Canadian Journal of Mathematics 41 (1989) 961–1004.
[29] B. A. Davey, H. A. Priestley, Introduction to lattices and order, 2nd ed., Cambridge University Press, 2002.
[30] J. Meseguer, Twenty years of rewriting logic, J. Log. Algebraic Methods Program. 81 (2012) 721–781.
[31] L. Herman, E. Marsden, R. Piziak, Implication connectives in orthomodular lattices., Notre Dame Journal of Formal Logic 16 (1975) 305–328.
[32] M. Navara, On generating finite orthomodular sublattices, Tatra Mountains Mathematical Publications 10 (1997) 109–117.

Theoretical Foundation for Equivalence Checking of Quantum Circuits

Canh Minh Do*, Kazuhiro Ogata

Japan Advanced Institute of Science and Technology, 1-8 Asahidai, Nomi, Japan

Abstract

Quantum circuits are typically used to design quantum algorithms at a high abstraction level without considering specific hardware restrictions. To execute quantum circuits on an actual quantum device, they must undergo a compilation process, transforming the high abstraction level into a low abstraction level that conforms to all restrictions imposed on the target device. As a result, the original quantum circuits and their compiled counterparts differ significantly. Therefore, it is crucial to verify the equivalence of two quantum circuits constructed from quantum gates based on their functionality. This paper presents a theoretical foundation for checking the equivalence of quantum circuits based on which an algorithm is constructed. The equivalence of quantum circuits can be reduced to matrix equivalence modulo a global phase. To achieve this, we compare each column vector of two matrices modulo the same global phase, making it significantly faster than the actual matrix equivalence check, especially in cases of non-equivalent quantum circuits.

Keywords

Observable Equivalence, Density Matrices, Equivalence Checking, Quantum Circuits

1. Introduction

Quantum computing is a rapidly emerging technology that uses the principles of quantum mechanics to solve complex problems beyond the capabilities of current classical computing. Several quantum algorithms have been proposed, showing significant improvements over classical algorithms, such as Shor's fast algorithms for discrete logarithms and factoring in 1994 [1]. Although practical quantum computers capable of running such algorithms effectively are not yet available, recent exponential investments from big companies like IBM, Google, Microsoft, and Intel bring the future of the quantum era within closer reach.

Quantum circuits are a natural model of quantum computation, comprising qubits and quantum operations (e.g., quantum gates), that can be used to design and implement quantum algorithms. However, quantum circuits are typically used to design quantum algorithms at a high abstraction level without considering specific hardware restrictions. To execute the quantum circuits on an actual quantum device, they have to undergo a *compilation* process, transforming the high abstraction level to a low abstraction level that conforms to all restrictions

The 2nd International Workshop on Formal Analysis and Verification of Post-Quantum Cryptographic Protocols, November 11, 2023, Brisbane, Australia

*Corresponding author.

canhdo@jaist.ac.jp (C. M. Do); ogata@jaist.ac.jp (K. Ogata)

0000-0002-1601-4584 (C. M. Do); 0000-0002-4441-3259 (K. Ogata)

imposed on the targeted device. More precisely, this compilation process has several key aspects as follows. Firstly, quantum devices natively support only a limited set of quantum operations. Consequently, quantum circuits intended for the target device must be expressed using only these native quantum operations. This requires a *decomposition* (or *translation*) step of non-native quantum operations into sequences of native ones [2, 3, 4]. Secondly, logical qubits used in quantum circuits have to be mapped to physical qubits on the target device. However, this mapping cannot be arbitrary because the target device imposes restrictions on which physical qubits can interact with each other. To achieve this, a *mapping* (or *routing*) step is required, which involves adding SWAP and Hadmard gates to quantum circuits [5, 6, 7]. Lastly, after the decomposition and mapping steps, the size of quantum circuits tends to increase, posing challenges for their execution on quantum devices due to noise and decoherence effects. Therefore, an *optimization* step is required to reduce the size of quantum circuits in terms of the number of quantum gates [8, 9, 10]. As a result of these processes, the quantum circuit defined at a high abstraction level and its compiled counterpart defined at a low abstraction level are significantly different. Therefore, it is crucial to verify the equivalence of two quantum circuits based on their functionality.

The functionality of quantum circuits can be described by a sequence of quantum gates, which are represented by unitary matrices. Given two quantum circuits constructed from quantum gates in the form of $U = U_m \dots U_0$ and $U' = U'_{m'} \dots U'_0$, the two quantum circuits are considered equivalent if U is equal to U' modulo a global phase, which is physically unobservable [11]. Directly comparing U and U' is inefficient because it requires costly matrix-matrix multiplications $U_m \dots U_0$ and $U'_{m'} \dots U'_0$ to obtain U and U' for comparison. Moreover, if U is significantly different from U', constructing the entire elements of both matrices is unnecessary. Instead, we can compare each column of two matrices modulo a global phase. Concretely, for each basis vector $|\phi_i\rangle$ in an orthonormal basis of a Hilbert space, if $U|\phi_i\rangle$ is equal to $U'|\phi_i\rangle$ modulo the same global phase, then the two quantum circuits are equivalent. It is important to note that we may need a few iterations to check that some quantum circuits are not equivalent in cases of non-equivalent quantum circuits, making it significantly faster than the actual matrix equivalence check. We present a theoretical foundation for checking the equivalence of quantum circuits with a theorem to guarantee the correctness of our approach. An algorithm is also constructed based on our theorem.

The rest of the paper is organized as follows: Sect. 2 provides the basics of quantum mechanics related mostly to linear algebra, Sect. 3 proposes a theoretical foundation of equivalence checking of quantum circuits in this work based on which an algorithm is constructed, Sect. 4 presents some existing work, and Sect. 5 concludes the paper with some pieces of future work.

2. Basic Quantum Mechanics

In classical computing, the fundamental unit of information is a bit whose value is either 0 or 1. In quantum computing, the counterpart is a *quantum bit* or *qubit*, which has two basis states, conventionally written in Dirac notation [12] as $|\mathbf{0}\rangle$ and $|\mathbf{1}\rangle$, which denote two column vectors $\begin{pmatrix}1\\0\end{pmatrix}$ and $\begin{pmatrix}0\\1\end{pmatrix}$, respectively. In quantum theory, a general state of a quantum system

is a superposition or linear combination of basis states. A quantum state is a unit vector in a Hilbert space $\mathcal{H}$, which is a vector space equipped with an inner product such that each Cauchy sequence has a limit. The state of a single qubit is $|\psi\rangle = \alpha|\mathbf{0}\rangle + \beta|\mathbf{1}\rangle$, where α and β are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$. States can be represented by column complex vectors as follows: $|\psi\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \alpha|\mathbf{0}\rangle + \beta|\mathbf{1}\rangle$, where $\{|\mathbf{0}\rangle, |\mathbf{1}\rangle\}$ forms an orthonormal basis of the two-dimensional complex vector space.

The basis $\{|\mathbf{0}\rangle, |\mathbf{1}\rangle\}$ is called as the *standard* basis. Besides, there are some other orthonormal bases studied in the literature, such as *diagonal* (or *dual*, or *Hadamard*) basis consisting of the following vectors:

$$|+\rangle = \tfrac{1}{\sqrt{2}}(|\mathbf{0}\rangle + |\mathbf{1}\rangle) \text{ and } |-\rangle = \tfrac{1}{\sqrt{2}}(|\mathbf{0}\rangle - |\mathbf{1}\rangle)$$

The evolution of a closed quantum system can be performed by a unitary transformation. If the state of a qubit is represented by a column vector, then a unitary transformation $\boldsymbol{U}$ can be represented by a complex-value matrix such that $\boldsymbol{U}\boldsymbol{U}^\dagger = \boldsymbol{U}^\dagger\boldsymbol{U} = \boldsymbol{I}$ or $\boldsymbol{U}^\dagger = \boldsymbol{U}^{-1}$, where $\boldsymbol{U}^\dagger$ is the conjugate transpose of $\boldsymbol{U}$. $\boldsymbol{U}$ acts on the Hilbert space $\mathcal{H}$ transforming a state $|\psi\rangle$ to a state $|\psi'\rangle$ by a matrix multiplication such that $|\psi'\rangle = \boldsymbol{U}|\psi\rangle$. There are some frequently used quantum gates in applications: the Hadamard gate $\boldsymbol{H}$, the identity gate $\boldsymbol{I}$, the Pauli gates $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$, and the controlled-NOT gate $\boldsymbol{CX}$. Note that the $\boldsymbol{CX}$ gate performs on two qubits, while the remaining gates perform on a single qubit. Their matrix representations are as follows:

$$\boldsymbol{I}_2 = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}, \qquad \boldsymbol{X} = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}, \qquad \boldsymbol{Y} = \begin{pmatrix}0 & -i\\ i & 0\end{pmatrix},$$

$$\boldsymbol{Z} = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}, \qquad \boldsymbol{H} = \tfrac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}, \qquad \boldsymbol{CX} = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}.$$

where i is the imaginary unit. For example, the Hadamard gate on a single qubit performs the mapping $|\mathbf{0}\rangle \mapsto \frac{1}{\sqrt{2}}(|\mathbf{0}\rangle + |\mathbf{1}\rangle)$ and $|\mathbf{1}\rangle \mapsto \frac{1}{\sqrt{2}}(|\mathbf{0}\rangle - |\mathbf{1}\rangle)$. The controlled-NOT gate on pairs of qubits performs the mapping $|\mathbf{00}\rangle \mapsto |\mathbf{00}\rangle, |\mathbf{01}\rangle \mapsto |\mathbf{01}\rangle, |\mathbf{10}\rangle \mapsto |\mathbf{11}\rangle, |\mathbf{11}\rangle \mapsto |\mathbf{10}\rangle$, which can be understood as inverting the second qubit (referred to as the *target*) if and only if the first qubit (referred to as the *control*) is one.

For multiple qubits, we use the tensor product of Hilbert spaces. Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be two Hilbert spaces. Their tensor product $\mathcal{H}_1 \otimes \mathcal{H}_2$ is defined as a vector space consisting of linear combinations of the vectors $|\psi_1\psi_2\rangle = |\psi_1\rangle|\psi_2\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$, where $|\psi_1\rangle \in \mathcal{H}_1$ and $|\psi_2\rangle \in \mathcal{H}_2$. Systems of two or more qubits may be in *entangled* states, meaning that states of qubits are correlated and inseparable. Entanglement shows that an entangled state of two qubits cannot be expressed as a tensor product of single-qubit states. We can use $\boldsymbol{H}$ and $\boldsymbol{CX}$ gates to create entangled states as follows: $\boldsymbol{CX}((\boldsymbol{H} \otimes \boldsymbol{I})|\mathbf{00}\rangle) = \frac{1}{\sqrt{2}}(|\mathbf{00}\rangle + |\mathbf{11}\rangle)$.

Let $|\psi\rangle$ be a quantum state and $\theta \in [0, 2\pi)$. In quantum mechanics, the state $e^{i\theta}|\psi\rangle$ is considered to be physically equal to $|\psi\rangle$ with respect to the global phase factor $e^{i\theta}$. From an observable perspective, two states are undistinguishable if they differ only by a global phase. We can use density matrices $|\psi\rangle\langle\psi|$ to present quantum states $|\psi\rangle$ from which we can eliminate the global phase factor as follows: $e^{i\theta}|\psi\rangle (e^{i\theta}|\psi\rangle)^\dagger = e^{i\theta}|\psi\rangle e^{-i\theta}\langle\psi| = |\psi\rangle\langle\psi|$.

3. Equivalence Checking of Quantum Circuits

This section presents the theoretical foundation of equivalence checking of quantum circuits by introducing a theorem based on which an algorithm is constructed.

3.1. Theoretical Foundation

We propose a method for checking the equivalence of quantum circuits constructed from quantum gates based on their functionality. We suppose that quantum circuits operate on quantum states in a Hilbert space $\mathcal{H}$ with n qubits. The unitary evolution of quantum systems is described by unitary matrices whose size is $2^n \times 2^n$. We first define the equivalence checking problem of quantum circuits.

Definition 3.1 (Equivalence checking problem)**.** Given two quantum circuits represented by unitary matrices, $U = U_m \dots U_0$ and $U' = U'_{m'} \dots U'_0$, the equivalence checking problem of U and U' is asked to check whether $U = e^{i\theta}U'$ for some $\theta \in [0, 2\pi)$.

We call $e^{i\theta}$ a global phase that is physically unobservable [11]. In quantum mechanics, quantum states that differ only by a global phase are physically indistinguishable and equivalent under observation [11]. Hence, we define observable equivalence for quantum states as follows:

Definition 3.2 (Observable equivalence for quantum states)**.** $|\psi\rangle \approx |\psi'\rangle$ (or $|\psi\rangle \approx_\theta |\psi'\rangle$ to make it clear from the context) is defined as $|\psi\rangle = e^{i\theta}|\psi'\rangle$ for some $\theta \in [0, 2\pi)$.

To check $|\psi\rangle \approx |\psi'\rangle$, we can check the equality of their density matrices $|\psi\rangle\langle\psi|$ and $|\psi'\rangle\langle\psi'|$. Because using density matrices to represent quantum states can eliminate the global phase, it is a handy trick to check the observable equivalence of two quantum states by comparing their density matrices. This result is derived from the following lemma.

Lemma 3.1. $|\psi\rangle \approx |\psi'\rangle$ if and only if $|\psi\rangle\langle\psi| = |\psi'\rangle\langle\psi'|$.

Proof. For the '*only if*' part (the $\Rightarrow$ direction), it is straightforward.

We now consider the '*if*' part (the $\Leftarrow$ direction). Let $\{|\phi_0\rangle, \dots, |\phi_i\rangle, \dots |\phi_{2^n-1}\rangle\}$ be an orthonormal basis of $\mathcal{H}$ with n dimension. There exists $c_0, \dots, c_i, \dots, c_{2^n-1}$ such that $|\psi\rangle = c_0|\phi_0\rangle + \dots + c_i|\phi_i\rangle + \dots + c_{2^n-1}|\phi_{2^n-1}\rangle$ and $|c_0|^2 + \dots + |c_i|^2 + \dots + |c_{2^n-1}|^2 = 1$. Similarly, we have $|\psi'\rangle = c'_0|\phi_0\rangle + \dots + c'_i|\phi_i\rangle + \dots + c'_{2^n-1}|\phi_{2^n-1}\rangle$. Let A and A' be $2^n \times 2^n$ matrices denoting $|\psi\rangle\langle\psi|$ and $|\psi'\rangle\langle\psi'|$, respectively. The elements of matrix A are calculated as follows:

$$\begin{aligned} a_{ij} &= \langle\phi_i| A |\phi_j\rangle \\ &= \langle\phi_i|\psi\rangle \langle\psi|\phi_j\rangle \end{aligned}$$

$$= c_i c_j^*$$

Similarly, the elements of matrix A' are calculated as follows: $a'_{ij} = c'_i c'^*_j$.

We have $a_{ij} = a'_{ij}$ for any $i, j \in [0 \dots 2^n - 1]$ because of A = A' from the assumption. Let us consider $a_{ii} = a'_{ii}$ as follows:

$$\begin{aligned}
& a_{ii} = a'_{ii} \\
\Leftrightarrow\ & c_i c_i^* = c'_i c'^*_i \\
\Leftrightarrow\ & re^{i\alpha}(r.e^{i\alpha})^* = r'e^{i\alpha'}(r'e^{i\alpha'})^* && \text{(by the exponential form of complex numbers)} \\
\Leftrightarrow\ & r = r' && \text{(because r and r' are non-negative numbers)}
\end{aligned}$$

We have $c_i = re^{i\alpha}$, $c'_i = r'e^{i\alpha'}$, and $r = r'$, where $i \in [0 \dots 2^n - 1]$. Let us consider two cases:

- If $r = r' = 0$, then it is immediate that $c_i = e^{i\theta_i} c'_i$ for some $\theta_i \in [0, 2\pi)$.
- If $r = r' \neq 0$, then we have $\frac{c_i}{c'_i} = \frac{re^{i\alpha}}{r'e^{i\alpha'}} = e^{i(\alpha - \alpha')} = e^{i\theta_i}$, where $\theta_i = \alpha - \alpha'$. Then, we have $c_i = e^{i\theta_i} c'_i$ for some $\theta_i \in [0, 2\pi)$.

Therefore, for all $i \in [0 \dots 2^n - 1]$, there exists $\theta_i \in [0, 2\pi)$ such that $c_i = e^{i\theta_i} c'_i$. Now let us consider $a_{ij} = a'_{ij}$ as follows:

$$\begin{aligned}
& a_{ij} = a'_{ij} \\
\Leftrightarrow\ & c_i c_j^* = c'_i c'^*_j \\
\Leftrightarrow\ & e^{i\theta_i} c'_i (e^{i\theta_j} c'_j)^* = c'_i c'^*_j && \text{(by the result above)} \\
\Leftrightarrow\ & e^{i(\theta_i - \theta_j)} c'_i c'^*_j = c'_i c'^*_j
\end{aligned}$$

Therefore, we have $\theta_i = \theta_j$ for any $c_i, c_j \neq 0$. It indicates that $|\psi\rangle = e^{i\theta} |\psi'\rangle$ for some $\theta \in [0, 2\pi)$. From Definition 3.2, we have $|\psi\rangle \approx |\psi'\rangle$. □

Recall to check the equivalence of quantum circuits U and U', we need to check whether $U = e^{i\theta} U'$ for some $\theta \in [0, 2\pi)$. We can use the following lemma to solve this problem.

Lemma 3.2. Let U and U' be $2^n \times 2^n$ unitary matrices, then $U = e^{i\theta} U'$ for some $\theta \in [0, 2\pi)$ if and only if $U |\psi\rangle \approx_\theta U' |\psi\rangle$ for any vector $|\psi\rangle \in \mathcal{H}$.

Proof. For the '*only if*' part (the $\Rightarrow$ direction), for any $|\psi\rangle$, we have $U |\psi\rangle = e^{i\theta} U' |\psi\rangle$ for some $\theta \in [0, 2\pi)$ using the assumption. From Definition 3.2, we have $e^{i\theta} U' |\psi\rangle \approx U' |\psi\rangle$. Therefore, $U |\psi\rangle \approx_\theta U' |\psi\rangle$.

For the '*if*' part (the $\Leftarrow$ direction), because $U |\psi\rangle \approx_\theta U' |\psi\rangle$ for any $|\psi\rangle$ from the assumption, we have $U |\psi\rangle = e^{i\theta} U' |\psi\rangle$ for some $\theta \in [0, 2\pi)$. Therefore, $U = e^{i\theta} U'$ for some $\theta \in [0, 2\pi)$. □

In Lemma 3.2, it is unfeasible to consider any vector $|\psi\rangle \in \mathcal{H}$ because there are infinite vectors in $\mathcal{H}$. Therefore, we introduce the following lemma to help us check whether $U = e^{i\theta}U'$ by considering only basis vectors in an orthonormal basis of $\mathcal{H}$. If the dimension of $\mathcal{H}$ is n, we need to consider at most 2^n basis vectors with respect to the same global phase.

Lemma 3.3. Let U and U' be $2^n \times 2^n$ matrices, then $U = e^{i\theta}U'$ for some $\theta \in [0, 2\pi)$ if and only if $U|\phi_i\rangle \approx_\theta U'|\phi_i\rangle$ for each basis vector $|\phi_i\rangle$ in an orthonormal basis of $\mathcal{H}$.

Proof. The '*only if*' part (the $\Rightarrow$ direction) is immediate from Lemma 3.2.

For the '*if*' part (the $\Leftarrow$ direction), let $\{|\phi_0\rangle, \ldots, |\phi_i\rangle, \ldots |\phi_{2^n-1}\rangle\}$ be an orthonormal basis of $\mathcal{H}$ with n dimension. For each basis vector ϕ_i, we have $U|\phi_i\rangle \approx_\theta U'|\phi_i\rangle$ from the assumption. From Definition 3.2, we have $U|\phi_i\rangle = e^{i\theta}U'|\phi_i\rangle$ for some $\theta \in [0, 2\pi)$. Therefore, for any complex number c_i, we have $Uc_i|\phi_i\rangle = e^{i\theta}U'c_i|\phi_i\rangle$ (1).

For arbitrary $|\psi\rangle \in \mathcal{H}$, there exists $c_0, \ldots, c_i, \ldots, c_{2^n-1}$ such that $|\psi\rangle = c_0|\phi_0\rangle + \cdots + c_i|\phi_i\rangle + \cdots + c_{2^n-1}|\phi_{2^n-1}\rangle$ and $|c_0|^2 + \cdots + |c_i|^2 + \cdots + |c_{2^n-1}|^2 = 1$. Because of (1), we have the following:

$$\begin{aligned}
U|\psi\rangle &= Uc_0|\phi_0\rangle + \cdots + Uc_i|\phi_i\rangle + \cdots + Uc_{2^n-1}|\phi_{2^n-1}\rangle \\
&= e^{i\theta}U'c_0|\phi_0\rangle + \cdots + e^{i\theta}U'c_i|\phi_i\rangle + \cdots + e^{i\theta}U'c_{2^n-1}|\phi_{2^n-1}\rangle \\
&= e^{i\theta}U'(c_0|\phi_0\rangle + \cdots + c_i|\phi_i\rangle + \cdots + c_{2^n-1}|\phi_{2^n-1}\rangle) \\
&= e^{i\theta}U'|\psi\rangle
\end{aligned}$$

for any $|\psi\rangle$ and θ. It means that $U|\psi\rangle \approx_\theta U'|\psi\rangle$ for any vector $|\psi\rangle$. Therefore, $U = e^{i\theta}U'$ by Lemma 3.2. □

Remark 3.1. Checking $U|\psi_i\rangle \approx_\theta U'|\psi_i\rangle$ is actually checking the observable equivalence of the i-th column vector of U and the i-th column vector of U' with respect to the phase θ.

Lemma 3.4. $U|\phi_i\rangle \approx_\theta U'|\phi_i\rangle$ for each basis vector $|\phi_i\rangle$ in an orthonormal basis of $\mathcal{H}$ if and only if $U|\phi_i\rangle \approx U'|\phi_i\rangle$ for each $|\phi_i\rangle$ and $U|\phi_i\rangle (U|\phi_j\rangle)^\dagger = U'|\phi_i\rangle (U'|\phi_j\rangle)^\dagger$ for each $|\phi_i\rangle$ and $|\phi_j\rangle$.

Proof. For the '*only if*' part (the $\Rightarrow$ direction), we have $U|\phi_i\rangle \approx U'|\phi_i\rangle$ for each basis vector $|\phi_i\rangle$ using the assumption. Moreover, we have $U|\phi_i\rangle = e^{i\theta}U'|\phi_i\rangle$ and $U|\phi_j\rangle = e^{i\theta}U'|\phi_j\rangle$ for each $|\phi_i\rangle$ and $|\phi_j\rangle$. Therefore, $U|\phi_i\rangle (U|\phi_j\rangle)^\dagger = e^{i\theta}U'|\phi_i\rangle (e^{i\theta}U'|\phi_j\rangle)^\dagger = U'|\phi_i\rangle (U'|\phi_j\rangle)^\dagger$

For the '*if*' part (the $\Leftarrow$ direction), we have $U|\phi_i\rangle = e^{i\theta_i}U'|\phi_i\rangle$ and $U|\phi_j\rangle = e^{i\theta_j}U'|\phi_j\rangle$ for each $|\phi_i\rangle$ and $|\phi_j\rangle$ using the first condition in the assumption. For the second condition in the assumption, we have as follows:

$$\begin{aligned}
& U|\phi_i\rangle (U|\phi_j\rangle)^\dagger = U'|\phi_i\rangle (U'|\phi_j\rangle)^\dagger \\
\Leftrightarrow\ & e^{i\theta_i}U'|\phi_i\rangle (e^{i\theta_j}U'|\phi_j\rangle)^\dagger = U'|\phi_i\rangle (U'|\phi_j\rangle)^\dagger \\
\Leftrightarrow\ & e^{i(\theta_i-\theta_j)}U'|\phi_i\rangle (U'|\phi_j\rangle)^\dagger = U'|\phi_i\rangle (U'|\phi_j\rangle)^\dagger
\end{aligned}$$

Therefore, we have $\theta_i = \theta_j$ for each $|\phi_i\rangle$ and $|\phi_j\rangle$. It indicates that $U|\phi_i\rangle \approx_\theta U'|\phi_i\rangle$ for each basis vector $|\phi_i\rangle$. □

Algorithm 1: Equivalence Checking of Quantum Circuits

input : n – the dimension of a Hilbert space
$U = U_m \dots U_0$ and $U' = U'_{m'} \dots U'_0$ – two quantum circuits
$\{|\phi_0\rangle, \dots, |\phi_{2^n-1}\rangle\}$ – an orthonormal basis of a Hilbert space $\mathcal{H}$
$\theta \in [0, 2\pi)$ – the phase

output : True ($U = e^{i\theta}U'$) or False ($U \neq e^{i\theta}U'$)

1 **forall** $|\phi_i\rangle \in \{|\phi_0\rangle, \dots, |\phi_{2^n-1}\rangle\}$ **do**
2 $\quad |u_i\rangle = U_m \cdot (\dots (U_0 \cdot |\phi_i\rangle) \dots)$
3 $\quad |u'_i\rangle = U'_{m'} \cdot (\dots (U'_0 \cdot |\phi_i\rangle) \dots)$
4 $\quad$ **if** $|u_i\rangle\langle u_i| \neq |u'_i\rangle\langle u'_i|$ **then**
5 $\quad\quad$ **return** False
6 $\quad$ **if** $i \neq 0 \wedge |u_0\rangle\langle u_i| \neq |u'_0\rangle\langle u'_i|$ **then**
7 $\quad\quad$ **return** False
8 **return** True

Based on Lemma 3.1, Lemma 3.3 and Lemma 3.4, we introduce the following theorem to check whether $U = e^{i\theta}U'$ for some $\theta \in [0, 2\pi)$.

Theorem 3.5. Let U and U' be $2^n \times 2^n$ matrices, then $U = e^{i\theta}U'$ for some $\theta \in [0, 2\pi)$ if and only if $U |\phi_i\rangle (U |\phi_i\rangle)^\dagger = U' |\phi_i\rangle (U' |\phi_i\rangle)^\dagger$ for each basis vector ϕ_i and $U |\phi_i\rangle (U |\phi_j\rangle)^\dagger = U' |\phi_i\rangle (U' |\phi_j\rangle)^\dagger$ for each $|\phi_i\rangle$ and $|\phi_j\rangle$ in an orthonormal basis of $\mathcal{H}$.

Proof. It is immediate from Lemma 3.1, Lemma 3.3 and Lemma 3.4. □

It is extremely expensive to calculate matrix-matrix multiplications $U_m \dots U_0$ and $U'_{m'} \dots U'_0$ to obtain U and U' and subsequently multiple with each $|\phi_i\rangle$ and $|\phi_j\rangle$ in Theorem 3.5 because of the exponential size of unitary matrices. Instead, we can perform a series of matrix-vector multiplications between unitary matrices and vectors in sequence as follows:

$$\left|u_i^0\right\rangle = U_0 |\phi_i\rangle, \left|u_i^1\right\rangle = U_1 \left|u_i^0\right\rangle, \dots, |u_i^m\rangle = U_m \cdot u_i^{m-1}$$

The i-th column vector of matrix U is $|u_i\rangle$ (i.e., $|u_i^m\rangle$) and similarly for the i-th column vector $|u'_i\rangle$ of matrix U'. We are now ready to check whether $|u_i\rangle\langle u_i|$ is equal to $|u'_i\rangle\langle u'_i|$ for the first condition in Theorem 3.5. Moreover, for the second condition in Theorem 3.5, it suffices to fix $|u_i\rangle$ and $|u'_i\rangle$, and check whether $|u_i\rangle\langle u_j| = \left|u'_i\right\rangle\left\langle u'_j\right|$ for all $j \neq i$. This is an efficient way to handle the calculation in Theorem 3.5.

3.2. Algorithm for Equivalence Checking of Quantum Circuits

An algorithm for equivalence checking of quantum circuits can be constructed based on Theorem 3.5, which is shown as Algorithm 1. Given two quantum circuits in the form of $U = U_m \dots U_0$ and $U' = U'_{m'} \dots U'_0$, and an orthonormal basis $\{|\phi_0\rangle, \dots, |\phi_{2^n-1}\rangle\}$, for each basis vector ϕ_i, we first construct the series of matrix-vector multiplications between unitary matrices and vectors to obtain $|u_i\rangle$ and $|u'_i\rangle$ in the code fragment at lines 2–3. We then check

whether their corresponding density matrices $|u_i\rangle\langle u_i|$ and $|u'_i\rangle\langle u'_i|$ are equal for the first condition in Theorem 3.5 in the code fragment at lines 4–5. If this is not the case, False is returned. Otherwise, we keep on checking for the second condition in Theorem 3.5 except for the case of the basis vector $|\phi_0\rangle$ in the code fragment at lines 6–7. If $|u_0\rangle\langle u_i|$ is not equal to $|u'_0\rangle\langle u'_i|$, False is returned. Otherwise, we move to check for other basis vectors. True is returned at the end once all basis vectors have been checked.

4. Related Work

L. Burgholzer et al. [13] have proposed an advanced method for equivalence checking of quantum circuits based on a decision diagram. Their approach involves two quantum circuits $U = U_m \dots U_0$ and $U'_{m'} \dots U'_0$ as inputs and they check whether the two quantum circuits are equivalent. They leverage two key observations: (1) quantum circuits are inherently reversible, and (2) even small differences in quantum circuits may impact the overall behavior of quantum circuits. Their strategy is as follows. For (2), they first randomly prepare some basis vectors ϕ_i and calculate the i-th column of each matrix U and U' to obtain $|u_i\rangle$ and $|u'_i\rangle$ as we do. They then compare $|u_i\rangle$ and $|u'_i\rangle$ modulo the global phase by using the fidelity, denoted $\mathcal{F} = |\langle u_i|u'_i\rangle|^2$, to measure the overlap between the two states. The two states are considered equivalent if the fidelity between them is 1 up to a given tolerance ε. This is an approximate estimation, while we use their density matrices for the comparison, which provides an exact estimation. After several runs, if they find $|u_i\rangle$ and $|u'_i\rangle$ that are not equivalent, the process is stopped. Otherwise, they attempt to resolve $U \Rightarrow I \Leftarrow U'$ into the identity matrix I based on (1) to solve the equivalence checking problem. However, calculating $U_i(U'_{i'})^\dagger$ involves an expensive matrix-matrix multiplication. Moreover, they seem not to consider the global phase in this step. We have proven a theorem that it suffices to consider all basis vectors in an orthonormal basis with the same global phase to conclude the equivalence checking problem. Our approach considers the global phase and can be adopted by other approaches/tools for checking the equivalence of quantum circuits.

5. Conclusion

We have presented a theoretical foundation for checking the equivalence of quantum circuits based on which an algorithm is constructed. The equivalence checking process is simplified to comparing each column vector of two unitary matrices, representing two quantum circuits, modulo the same global phase. To eliminate the global phase during the comparison, we compare their corresponding density matrices instead of their column vectors. To guarantee the same global phase is used, we compare the outer products of two columns from each unitary matrix, with one column fixed. We have proven a theorem to guarantee the correctness of our approach. As one piece of future work, we would develop a support tool based on symbolic reasoning in [14, 15] for our approach and conduct case studies to demonstrate the effectiveness of our approach for equivalence checking of quantum circuits.

Acknowledgments

The research was supported by JAIST Research Grant for Fundamental Research, by JST SICORP Grant Number JPMJSC20C2, and by JSPS KAKENHI Grant Number JP23K19959.

References

[1] P. Shor, Algorithms for quantum computation: discrete logarithms and factoring, in: Proceedings 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 124–134. doi:10.1109/SFCS.1994.365700.

[2] B. Giles, P. Selinger, Exact synthesis of multiqubit clifford+t circuits, Phys. Rev. A 87 (2013) 032332. doi:10.1103/PhysRevA.87.032332.

[3] M. Amy, D. Maslov, M. Mosca, M. Roetteler, A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits, Trans. Comp.-Aided Des. Integ. Cir. Sys. 32 (2013) 818–830. doi:10.1109/TCAD.2013.2244643.

[4] D. Maslov, Advantages of using relative-phase toffoli gates with an application to multiple control toffoli optimization, Physical Review A 93 (2016). doi:10.1103/physreva.93.022311.

[5] M. Y. Siraichi, V. F. d. Santos, C. Collange, F. M. Q. Pereira, Qubit allocation, in: Proceedings of the 2018 International Symposium on Code Generation and Optimization, CGO 2018, Association for Computing Machinery, New York, NY, USA, 2018, p. 113–125. doi:10.1145/3168822.

[6] A. Zulehner, A. Paler, R. Wille, Efficient mapping of quantum circuits to the ibm qx architectures, in: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018, pp. 1135–1138. doi:10.23919/DATE.2018.8342181.

[7] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, S. Sivarajah, On the qubit routing problem (2019). doi:10.4230/LIPICS.TQC.2019.5.

[8] D. Maslov, G. W. Dueck, D. M. Miller, C. Negrevergne, Quantum circuit simplification and level compaction, Trans. Comp.-Aided Des. Integ. Cir. Sys. 27 (2008) 436–444. doi:10.1109/TCAD.2007.911334.

[9] Z. Sasanian, D. M. Miller, Reversible and quantum circuit optimization: A functional approach, in: R. Glück, T. Yokoyama (Eds.), Reversible Computation, Springer Berlin Heidelberg, 2013, pp. 112–124. doi:10.1007/978-3-642-36315-3_9.

[10] Y. Nam, N. J. Ross, Y. Su, A. M. Childs, D. Maslov, Automated optimization of large quantum circuits with continuous parameters, npj Quantum Information 4 (2018) 23. doi:10.1038/s41534-018-0072-4.

[11] M. A. Nielsen, I. L. Chuang, Quantum Computation and Quantum Information: 10th Anniversary Edition, Cambridge University Press, 2010. doi:10.1017/CBO9780511976667.

[12] P. A. M. Dirac, A new notation for quantum mechanics, Mathematical Proceedings of the Cambridge Philosophical Society 35 (1939) 416–418. doi:10.1017/S0305004100021162.

[13] L. Burgholzer, R. Wille, Advanced equivalence checking for quantum circuits, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 40 (2021) 1810–1824. doi:10.1109/TCAD.2020.3032630.

[14] C. M. Do, K. Ogata, Symbolic model checking quantum circuits in maude, in: The 35th International Conference on Software Engineering and Knowledge Engineering, SEKE 2023, 2023, pp. 103–108. doi:10.18293/SEKE2023-014.
[15] T. Takagi, C. M. Do, K. Ogata, Automated quantum program verification in dynamic quantum logic (to appear), in: DaLí: Dynamic Logic – New trends and applications, 2023.