symQV: Automated Symbolic Verification of Quantum Programs

Fabian Bauer-Marquart[1](✉), Stefan Leue[1], and Christian Schilling[2]

[1] University of Konstanz, Konstanz, Germany
fabian@bauer-marquart.com, stefan.leue@uni-konstanz.de
[2] Aalborg University, Aalborg, Denmark
christianms@cs.aau.dk

Abstract. We present symQV, a symbolic execution framework for writing and verifying quantum computations in the quantum circuit model. symQV can automatically verify that a quantum program complies with a first-order specification. We formally introduce a symbolic quantum program model. This allows to encode the verification problem in an SMT formula, which can then be checked with a δ-complete decision procedure. We also propose an abstraction technique to speed up the verification process. Experimental results show that the abstraction improves symQV's scalability by an order of magnitude to quantum programs with 24 qubits (a 2^{24}-dimensional state space).

Keywords: Quantum computing · Formal verification · Symbolic execution · Abstraction

1 Introduction

Quantum computing bears great potential in increasing the scalability of problem solving in many areas such as optimization [15,25], database search [19], cryptography [36], quantum dynamics simulation [10], satisfiability problems [8], and machine learning [23]. Recently, quantum computing has gained momentum with applications in safety-critical domains such as traffic flow [18], aircraft load [38], logistics [2], and medical diagnostics [21]. Furthermore, quantum simulation [1,11,37] and quantum computers in the cloud [22] are now available.

As with classical programs, detecting bugs in quantum programs is a crucial problem. For classical programs, there exist powerful formal verification techniques to automatically verify that the programs comply with a formal specification [12]. State-of-the-art verifiers, e.g., for C programs [6,7,27] perform verification *symbolically*: The developer marks specific program inputs as symbolic so that the verifier knows to use these as the "search space." The verifier then proves that all possible inputs to the program comply with the specification.

F. Bauer-Marquart—The work was done while the first author was employed at the University of Konstanz.

M. Chechik et al. (Eds.): FM 2023, LNCS 14000, pp. 181–198, 2023.
https://doi.org/10.1007/978-3-031-27481-7_12

For quantum programs, this level of automation is not yet available. In this work, we aim to bridge this gap. Existing approaches to quantum program analysis can be categorized in three directions:

Interactive Proof Assistants: Several approaches [9,20,29,30,33] propose using interactive proof assistants to verify quantum programs. These works provide a large set of deductions but require familiarity with proof assistants such as Coq [5] or Isabelle/HOL [32], competence in proof-writing, and many hours of manual programming work to conduct the verification. These techniques are not fully automatic, which would be crucial for keeping pace with the development of quantum algorithms [24].

Automated Quantum Compiler Verification: Amy [3] proposes an efficient path-sum framework that performs fully automated equivalence checking of a quantum program against a simpler version of the same program, as well as against path-sums that the author uses as specification. The approach is applicable to quantum programs written with quantum gates from the Clifford+T group. Shi et al. [35] use an SMT (satisfiability modulo theories) solver to verify a quantum compiler via equivalence checking. These approaches do not handle general formal specifications.

Quantum Assertion Checking: Li et al. [28] verify assertions during quantum program run-time via projections. Yu and Palsberg [39] use an abstraction to verify assertions on quantum programs with up to 300 qubits, but the approach is restricted to programs where inputs are fixed to a specific value. This is a severe drawback, as essential quantum algorithms such as teleportation, the quantum Fourier transform [31], or Grover's diffusion operator [19] require arbitrarily-valued inputs.

In summary, despite the significance of ensuring specification compliance in quantum software engineering, there is still a lack of practical, automated tools for the purpose of symbolic quantum verification of general formal specifications. Existing tools either:

- require a high amount of manual programming,
- restrict the type of quantum program, e.g., support only a subset of quantum gates or only measurement-free quantum programs,
- do not work symbolically, requiring to fix the inputs to the program, or
- do not support the checking of formal specifications written in first-order logic, which is the standard for classical software verification.

In this paper, we introduce symQV, a framework for writing and verifying quantum programs in the quantum circuit model. To the best of our knowledge, symQV is the first tool that allows automated "push-button" verification of quantum programs where the programs are executed symbolically. In *symbolic execution*, a program is not executed with a predetermined input value. Instead, it is executed with the complete range of possible input values. In contrast to the classical case, where the number of possible input values is bounded by the RAM architecture, the range of input values to a quantum program is infinite.

symQV's automation and high-level workflow are similar to classical verification frameworks such as CPAchecker [6]: quantum developers only need to write a quantum program (using a Cirq-like [11] syntax) and a first-order logic specification that expresses the desired program output. Then, compliance with this specification is automatically verified based on SMT technology. If the quantum program does not satisfy the specification, the user obtains a counterexample that aids in locating errors in the program.

A major obstacle in practice is that quantum program simulators require exponential memory in the number of qubits. This is because simulators running on classical computers need to utilize a matrix to represent the state of a quantum mechanical system. This matrix doubles in size with every qubit that is added to the computation [31], which naturally carries over to verifying quantum programs. We show that in many practical cases this exponential matrix representation can be avoided. In addition, we propose an *abstraction* (or *over-approximation*) [13] that makes our technique more scalable without harming verification soundness.

We evaluate our approach symQV on essential quantum algorithms and subroutines. These include teleportation, QFT, [31], Grover's diffusion operator [19], and quantum phase estimation [36]. We demonstrate that symQV efficiently verifies quantum programs with up to 24 symbolic input qubits (a 2^{24}-dimensional state space), showing its potential to be used as a general-purpose verifier by developers of quantum programs. To put this number into perspective: state-of-the-art quantum computers currently offer one error-corrected qubit [26].

The main contributions of this paper can be summarized as follows. **First**, we introduce a symbolic quantum program model to express quantum programs and safety specifications in our verification framework. **Second**, we provide an encoding of the quantum program model in SMT and show that this encoding is sound and complete. We use this encoding to automatically verify formal specifications written in first-order logic. **Third**, we introduce a sound abstraction technique, which improves the verification time by one order of magnitude. **Finally**, we evaluate our implementation symQV on several quantum programs with up to 24 qubits.

2 Background

This section briefly introduces the concepts of quantum computing used in this paper. For detailed explanations, we refer to Nielsen and Chuang [31].

The *qubit* is the basic unit of quantum information. A single qubit can be in the *ground state* $|0\rangle$ ("ket zero") or in the *excited state* $|1\rangle$ ("ket one"). In general, however, a qubit is in a superposition of both *computational basis states*, written as $|q\rangle = \alpha|0\rangle + \beta|1\rangle$. The *amplitudes* $\alpha, \beta \in \mathbb{C}$ characterize a qubit, with $|\alpha|^2$ and $|\beta|^2$ being the probability of the qubit to be in either state. Therefore, their values are restricted such that $|\alpha|^2 + |\beta|^2 = 1$. Qubits are often written as two-dimensional vectors:

$$|0\rangle \equiv \begin{bmatrix}1\\0\end{bmatrix}, \qquad |1\rangle \equiv \begin{bmatrix}0\\1\end{bmatrix}, \qquad |q\rangle \equiv \begin{bmatrix}\alpha\\\beta\end{bmatrix}.$$

The qubit states span a two-dimensional Hilbert space $\mathcal{H}_2 = \{\alpha\,|0\rangle + \beta\,|1\rangle\}$, a complete complex vector space where the inner product is defined. When we combine n qubits, the system's state vector $|\psi\rangle$ spans the *tensor product* of Hilbert spaces $\mathcal{H}_{2^n} = \bigotimes_{i=1}^{n} \mathcal{H}_2^{(i)}$, and $|\psi\rangle$ is a 2^n-dimensional vector.

Quantum logic gates are the building blocks of quantum programs and transform a quantum state into a new quantum state. They are characterized by unitary matrices U that transform quantum state vectors. Common quantum gates, shown in Fig. 1, include X (Not), Z (phase-flip), H (Hadamard), U_{CX} (controlled-Not), and U_{CZ} (controlled phase-flip).

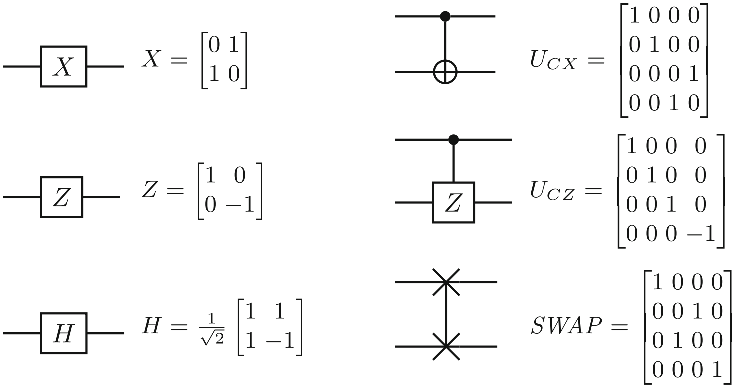

X
$X = \begin{bmatrix}0&1\\1&0\end{bmatrix}$
$U_{CX} = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}$
Z
$Z = \begin{bmatrix}1&0\\0&-1\end{bmatrix}$
Z
$U_{CZ} = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&-1\end{bmatrix}$
H
$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1&1\\1&-1\end{bmatrix}$
$SWAP = \begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}$

Fig. 1. Circuit diagrams and matrices of some common quantum gates. For the controlled gates U_{CX} and U_{CZ}, the dot (●) marks the control qubit.

The state of a qubit can alternatively be described with polar coordinates,

$$|q\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}\sin\frac{\theta}{2}\,|1\rangle\,,$$

where ϕ and θ correspond to angles that describe a point on the unit sphere, known as the *Bloch sphere* (see Fig. 2), with $|0\rangle$ being the north pole and $|1\rangle$ being the south pole. For instance, the gates X and Z perform a 180° rotation around the x and z axes, respectively, while H maps ground state $|0\rangle$ to $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ at the equator.

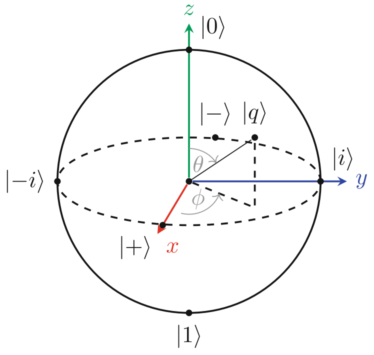
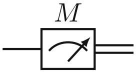


Fig. 2. A qubit $|q\rangle$ visualized on the Bloch sphere.

2.1 Entanglement

Quantum entanglement is an important concept of quantum mechanics. It occurs if the state of one qubit cannot be characterized independently of the state of another qubit, including when the qubits are separated over a large distance. Two-qubit states with perfect correlation are called the *Bell states*. An example for such a state is $|\phi^+\rangle = \frac{1}{\sqrt{2}} (|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle)$, where the first and second qubit are always guaranteed to be either both 0 or both 1 after measurement.

2.2 Quantum Measurement

Measuring a single qubit $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$ converts it into a classical bit: 0 with probability $|\alpha|^2$ and 1 with probability $|\beta|^2$. In circuit notation, a measurement is denoted as  (the double stroke indicates a *classical* wire). Because there are two statistical outcomes, 0 and 1, there exists one measurement operator (a non-unitary matrix) for each: $M_0 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ and $M_1 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$. The measurement operators irreversibly change the quantum state, which influences subsequent computations. Because of the statistical nature of quantum measurement, simulation tools (and also symQV) need to branch out into two execution paths, with a probability value associated with each of the paths.

2.3 Running Example: Teleportation

Quantum teleportation (TP) is an example of a quantum program with *symbolic* inputs; here, Alice wants to send a qubit $|\psi\rangle$ to Bob. There exists no quantum communication channel in this problem setting, but Alice and Bob each have one qubit of an entangled qubit pair $|\phi^+\rangle$. This is used to send (teleport) Alice's qubit to Bob: First, Alice uses a $CNOT$ and H gate to entangle her two qubits with each other. Then, after measuring both, she sends the measurement results via a classical communication channel to Bob, who finally retrieves $|\psi\rangle$ using two controlled gates, U_{CX} and U_{CZ}. The circuit diagram is shown in Fig. 3.

This example motivates the importance of *symbolic verification*: we want to verify that teleportation is successful for *any* quantum state and, hence, need to represent the input state symbolically.

3 The symQV Quantum Program Model

We introduce the *quantum program model* M_Q as an SMT-compatible symbolic representation of the general quantum circuit model [31]. The quantum program model, unlike the standard state-vector representation used in simulators, can represent operations on qubits as direct mappings in SMT instead of matrices. Only when necessary, for example when qubits become entangled, do we construct the state vector for this specific subset of qubits.

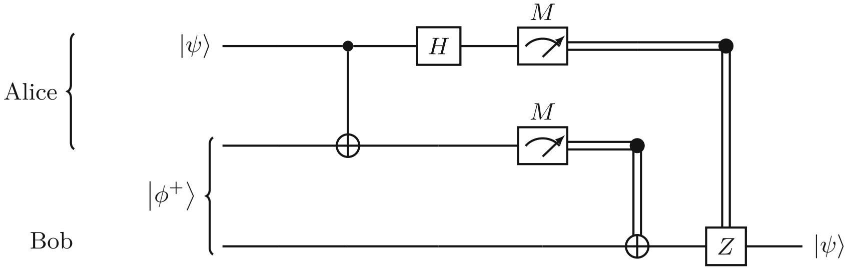


Fig. 3. Quantum teleportation circuit, adapted from [31]. The double line indicates a classical wire. Here, it simulates a communication channel.

The main benefit of the quantum program model is that it allows reasoning about quantum programs whose inputs are symbolic and therefore not fixed to a certain value. Thus we can use the model to perform formal verification against all possible inputs, i.e., the entire infinite Hilbert space. Furthermore, the quantum program model allows us to handle quantum programs with parametrized gates, which add another (infinite) dimension to the problem.

We give a high-level, bottom-up presentation of the quantum program model. At the end of the presentation we exemplify the encoding of the quantum teleportation program in Sect. 3.1 (the complete SMT formula is shown in Sect. A.4 of the supplementary material [4]). First, we need symbolic encodings for qubits, computations, and measurements. For convenience, we encode both the amplitudes and the phases into the qubit's SMT representation, allowing computations to work on either.

Encoding 1 (Qubit). *We encode a complex number as a pair $z := (z_R, z_I)$ with $z_R, z_I \in \mathbb{R}$. Using this representation, we encode a qubit as a 4-tuple*[1]

$$|q\rangle := (\alpha, \beta, \phi, \theta), \qquad \alpha, \phi, \theta \in \mathbb{R},\ \beta \in \mathbb{C}.$$

We combine both the amplitude and phase representation because we need to restrict the valuations of the variables using the following constraints:

$$\alpha = \cos\frac{\theta}{2} \ \wedge\ \beta_R = \cos\phi \cdot \sin\frac{\theta}{2} \ \wedge\ \beta_I = \sin\phi \cdot \sin\frac{\theta}{2}, \tag{1}$$

which constrains the qubit's degrees of freedom to $|\alpha|^2 + |\beta|^2 = 1$, and

$$0 \le \theta \le \pi \wedge 0 \le \phi < 2\pi \quad \wedge \quad \theta = 0 \Rightarrow \phi = 0 \quad \wedge \quad \theta = \pi \Rightarrow \phi = 0, \tag{2}$$

which constrains the angles' values to their respective periods.

Encoding 1 constrains a qubit's degree of freedom via its phases (Eq. (2)). This is because directly encoding the sphere equation $|\alpha|^2 + |\beta|^2 = 1$ requires two

[1] We choose α to be real because the global phase [31] has no observable consequences.

nested square operations, which are challenging for state-of-the-art SMT solvers (we evaluated Z3 [14] and dReal [17]).

The main motivation for our quantum program model is that we are often not required to build the whole (2^n-dimensional) state vector. Standard (unitary) quantum gates can be conveniently realized by a direct mapping on the SMT level, which we first define in an abstract way and instantiate later:

Definition 1 (Direct mapping). *We encode a unitary gate as a bijection* $U : \mathcal{H}_2^k \rightarrow \mathcal{H}_2^k$ *called* direct mapping*, where* k *is the number of modified qubits.*

Direct mappings allow us to express the effect of a quantum gate without explicitly constructing the matrix representation, unlike in standard quantum simulators. We concretize the notion of the direct mapping (Definition 1) with the following encodings of the most common quantum logic gates [31]:

Encoding 2.1 (Basic single-qubit gates). *The identity,* X*,* Z*, and* H *gates are encoded as the following mappings:*

$$I\left(\begin{bmatrix}\alpha\\ \beta\end{bmatrix}\right) := \begin{bmatrix}\alpha\\ \beta\end{bmatrix}, X\left(\begin{bmatrix}\alpha\\ \beta\end{bmatrix}\right) := \begin{bmatrix}\beta\\ \alpha\end{bmatrix}, Z\left(\begin{bmatrix}\alpha\\ \beta\end{bmatrix}\right) := \begin{bmatrix}\alpha\\ -\beta\end{bmatrix}, H\left(\begin{bmatrix}\alpha\\ \beta\end{bmatrix}\right) := \begin{bmatrix}\frac{\alpha+\beta}{\sqrt{2}}\\ \frac{\alpha-\beta}{\sqrt{2}}\end{bmatrix}.$$

We extend the encoding of the identity gate to take a variable number of arguments, such that $I(|q_0\rangle, \ldots, |q_k\rangle) = (|q_0\rangle, \ldots, |q_k\rangle)$ *for any* k*.*

The gates in Encoding 2.1 are used to modify the amplitudes of a qubit. The next encoding includes gates that modify a qubit's phases without directly affecting its amplitudes.

Encoding 2.2 (Phase gates). *The phase gates* R_X *and* R_Z *perform parametrized rotations around the* x *and* z *axes, respectively. The mappings use the phase angles:*

$$R_X(\theta')(\phi, \theta) := (\phi, \theta + \theta'), \qquad R_Z(\phi')(\phi, \theta) := (\phi + \phi', \theta).$$

Encoding 2.3 (*SWAP* gate). *The mapping of the SWAP gate applied to qubits* $|q_0\rangle$ *and* $|q_1\rangle$ *is*

$$SWAP(|q_0\rangle, |q_1\rangle) := (|q_1\rangle, |q_0\rangle).$$

In cases where it is not possible to express a quantum gate as a unitary mapping, such as entangling gates, we resort to the standard matrix representation. The matrix is then applied to a quantum state vector via matrix multiplication.

Encoding 3 (Gate matrix). *We encode a quantum gate as a* $2^k \times 2^k$ *(complex) matrix* U*, where* k *is the number of modified qubits. We further require that* U *is reversible (cf. Sect. 2).*

Encoding 4 (Matrix multiplication). *For an* $m \times n$ *matrix* A *and an* $n \times p$ *matrix* B*, the result of the matrix multiplication* $A \cdot B$*, an* $m \times p$ *matrix* C*, is encoded via the identities* $\bigwedge_{i=1}^{m} \bigwedge_{j=1}^{p} c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$*.*

There are benefits when encoding a gate via a direct mapping instead of a matrix, which we now illustrate with an example:

Example 1. Recall that the *SWAP* gate can be encoded via a direct mapping (Encoding 2.3), i.e., we can compute

$$SWAP(|q_0\rangle, |q_1\rangle) = (|q_1\rangle, |q_0\rangle)$$

in one step. This is *not* the case for the matrix encoding:

$$\begin{aligned} SWAP(|q_0\rangle \otimes |q_1\rangle) &= \begin{bmatrix} 1\,0\,0\,0 \\ 0\,0\,1\,0 \\ 0\,1\,0\,0 \\ 0\,0\,0\,1 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \beta_0 \end{bmatrix} \otimes \begin{bmatrix} \alpha_1 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} 1\,0\,0\,0 \\ 0\,0\,1\,0 \\ 0\,1\,0\,0 \\ 0\,0\,0\,1 \end{bmatrix} \begin{bmatrix} \alpha_0\alpha_1 \\ \alpha_0\beta_1 \\ \beta_0\alpha_1 \\ \beta_0\beta_1 \end{bmatrix} = \begin{bmatrix} \alpha_0\alpha_1 \\ \beta_0\alpha_1 \\ \alpha_0\beta_1 \\ \beta_0\beta_1 \end{bmatrix} \\ &= |q_1\rangle \otimes |q_0\rangle \,. \end{aligned}$$

Here we observe that the matrix representation is verbose. It needs 4 multiplications per tensor product and 16 multiplications only for computing the result of the matrix multiplication. Note that the number of operations increases exponentially with the number of qubits, illustrating the benefit of the direct mapping. We give a further example of a direct mapping in Sect. A.1 of the supplementary material [4].

Measurement, the only non-reversible operation in our encodings, assigns 0 or 1 to a qubit with a certain probability. For a state s consisting of a single qubit $|q\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, there are two possible subsequent states: $s'(0) = |0\rangle$ and $s'(1) = |1\rangle$. The probabilities $p(x)$ that state x occurs are

$$p(0) = |\alpha|^2, \; p(1) = |\beta|^2.$$

Therefore, for every quantum measurement taking place in $M_{\mathcal{Q}}$, in the case of non-zero probabilities $p(0)$ and $p(1)$, there are two possible successor states, one per measurement outcome.

Encoding 5 (Quantum measurement). *We encode the measurement operators by applying the standard measurement matrices (cf. Sect. 2) to Encodings 3 to 4.*

For entangled quantum states, qubits can no longer be characterized individually [31]. Therefore, our encoding cannot use the direct-mapping strategy from Definition 1 and we fall back to a vector representation of the quantum state.

Definition 2 (Modeling a quantum state). *We define a vector data structure to represent an n-qubit quantum state $|\psi\rangle$. This structure holds (cf. Sect. 2)* 2^n *(symbolic) complex numbers*

$$|\psi\rangle := (\alpha_1, \alpha_2, \cdots, \alpha_{2^n}).$$

Encoding 6 (Tensor product of matrices). *For an $m \times n$ matrix A and a $p \times q$ matrix B, the tensor product $A \otimes B$, an $(mp) \times (nq)$ matrix C, is encoded via equalities $\bigwedge_{i=1}^{m} \bigwedge_{k=1}^{p} \bigwedge_{j=1}^{n} \bigwedge_{l=1}^{q} c_{ik,jl} = a_{i,j} \cdot b_{k,l}$.*

The following encoding is needed for gate matrices that only apply to a subset of the qubits in the system. This is achieved by taking a tensor product with the identity matrix I.

Encoding 7 (Applying gates to a subset of qubits). *For a quantum state $|\psi\rangle$ over $n+1$ qubits and a quantum gate U over qubits $|q_i\rangle$ to $|q_j\rangle$ where $0 \leq i < j \leq n$, the next state is*

$$|\psi'\rangle = \begin{cases} I^{\otimes i-1} \otimes U \otimes I^{\otimes n-j} |\psi\rangle & \text{if } 0 < i, j < n, \\ U \otimes I^{\otimes n-j} |\psi\rangle & \text{if } 0 = i, j < n, \\ I^{\otimes i-1} \otimes U |\psi\rangle & \text{if } 0 < i, j = n, \\ U |\psi\rangle & \text{if } 0 = i, j = n. \end{cases}$$

Having assigned a logic representation to qubits, quantum gates, and quantum measurement, we can combine them to define the *quantum program model.*

Definition 3 (Quantum program model). *A* quantum program model *is a 5-tuple*

$$M_{\mathcal{Q}} := (\mathcal{Q}, S, \rightarrow, \Theta, V_0) \tag{3}$$

where

- *$\mathcal{Q}$ is a set of n (symbolic) qubits $\{|q_0\rangle, \ldots, |q_{n-1}\rangle\}$,*
- *S is a sequence of m (symbolic) states $(s_0, \ldots, s_{m-1})$,*
- *$\rightarrow$ is a sequence of $m-1$ state operations $(\rightarrow_1, \ldots, \rightarrow_{m-1})$,*
- *Θ is a set of (symbolic) parameters, and*
- *V_0 is the qubit initializer sequence.*

The qubits of $\mathcal{Q}$ are symbolic unless an initial valuation (assignment of a subset of qubits with concrete values) is provided in V_0. The initial state is $s_0 = (|q_{0,0}\rangle, \ldots, |q_{0,n-1}\rangle)$ and all following states $s_i \in S$ $(0 < i < m)$ again consist of symbolic qubits $(|q_{i,0}\rangle, \ldots, |q_{i,n-1}\rangle)$. Every state operation $\rightarrow_i$ is either

- a direct mapping (Definition 1); or
- a unitary matrix (Encoding 3); or
- a quantum measurement (Encoding 5).

We define the shorthand

$$s_{i-1} \rightarrow_i s_i = \begin{cases} \rightarrow_i (s_{i-1}) = s_i & \rightarrow_i \text{ is a direct mapping,} \\ \rightarrow_i \cdot \bigotimes_{j=0}^{n-1} |q_{(i-1,j)}\rangle = \bigotimes_{j=0}^{n-1} |q_{(i,j)}\rangle & \rightarrow_i \text{ is a matrix,} \end{cases}$$

and tie the states and operations together via $\bigwedge_{i=1}^{m-1} s_{i-1} \rightarrow_i s_i$.

A state operation can also be a quantum measurement M. When state s_{i-1} is measured, two possible subsequent states are created: $s_i(0)$ and $s_i(1)$ (Sect. 2.2). Additionally, we allow measurement of k qubits at the same time for a bit vector $x \in \{0,1\}^k$ such that M_x is the combined measurement.

The set Θ contains symbolic, real-valued variables that are used to parameterize state operations, e.g., rotations. The sequence $V_0 = (\Psi_0, \dots, \Psi_{n-1})$ contains sets of initial valuations $\Psi_i \subseteq \mathcal{H}_2$ (possibly singleton sets in case of a concrete valuation). The initial valuations are asserted to the initial qubits via $\bigwedge_{i=0}^{n-1} |q_i\rangle \in \Psi_i$.

Before we give an example, we note that the quantum program model $M_{\mathcal{Q}}$ is equivalent to the traditional presentation of quantum computing.

Theorem 1 (Equivalence). *The quantum program model $M_{\mathcal{Q}}$ (Definition 3) and the quantum circuit model [31] are equivalent.*

The proof for Theorem 1 is given in Sect. A.4 of the supplementary material [4].

3.1 Running Example: Quantum Program Model of Teleportation

Now that we have defined the quantum program model, we formalize our running example, teleportation, as $M_{\mathcal{Q}} = (\mathcal{Q}, S, \rightarrow, \varnothing, V_0)$, where

$$\begin{aligned}
\mathcal{Q} &= \{|q_0\rangle, |q_1\rangle, |q_2\rangle\},\\
S &= (s_0, s_1, s_2, s_3, s_4),\\
\rightarrow &= (U_{CX}(|q_0\rangle, |q_1\rangle), H(|q_0\rangle), \mathcal{M}(|q_0\rangle, |q_1\rangle), U_{CX}(|q_1\rangle, |q_2\rangle), U_{CZ}(|q_0\rangle, |q_2\rangle)),\\
V_0 &= (\mathcal{H}_2, \{|\phi^+\rangle\}).
\end{aligned}$$

Note that valuations V_0 are symbolic, so each input qubit can assume any state in the Hilbert space.

Next we provide a high-level encoding of this quantum program model in SMT. The complete SMT formula is shown in Sect. A of the supplementary material [4].

We begin by encoding the first state s_0, which contains the three input qubits $|q_{0,0}\rangle, |q_{0,1}\rangle, |q_{0,2}\rangle$. The first operation $s_0 \rightarrow_1 s_1$ is encoded as $|q_{1,0}\rangle \otimes |q_{1,1}\rangle = U_{CX}\,|q_{0,0}\rangle \otimes |q_{0,1}\rangle$, with s_1 containing the qubits $|q_{1,0}\rangle, |q_{1,1}\rangle, |q_{1,2}\rangle$ that encode the result of this operation. The remaining states and state operations are encoded as follows (we have omitted identity operations for the sake of brevity), with all entries connected with a conjunction:

We observe that the measurement step from s_2 to s_3 results in the creation of 4 possible execution paths, one per measurement outcome $(00, 01, 10, 11)$. Also, recall that all the symbols and operators used in the encoding above, such as the tensor product ($\otimes$), gates (H, U_{CX}, U_{CZ}), measurements (M_0, M_1), and Hilbert space ($\mathcal{H}_2$), carry the meanings we assigned to them in Encodings 1 to 7.

State	Operation
$s_2 = (\lvert q_{2,0}\rangle, \lvert q_{2,1}\rangle, \lvert q_{2,2}\rangle)$	$\lvert q_{2,0}\rangle = H\lvert q_{1,0}\rangle$
$s_3(00) = (\lvert q_{3,0}(00)\rangle, \lvert q_{3,1}(00)\rangle, \lvert q_{3,2}(00)\rangle)$	$\lvert q_{3,0}(00)\rangle = M_0\lvert q_{2,0}\rangle,\ \lvert q_{3,1}(00)\rangle = M_0\lvert q_{2,1}\rangle$
$s_3(01) = (\lvert q_{3,0}(01)\rangle, \lvert q_{3,1}(01)\rangle, \lvert q_{3,2}(01)\rangle)$	$\lvert q_{3,0}(01)\rangle = M_0\lvert q_{2,0}\rangle,\ \lvert q_{3,1}(01)\rangle = M_1\lvert q_{2,1}\rangle$
$s_3(10) = (\lvert q_{3,0}(10)\rangle, \lvert q_{3,1}(10)\rangle, \lvert q_{3,2}(10)\rangle)$	$\lvert q_{3,0}(10)\rangle = M_1\lvert q_{2,0}\rangle,\ \lvert q_{3,1}(10)\rangle = M_0\lvert q_{2,1}\rangle$
$s_3(11) = (\lvert q_{3,0}(11)\rangle, \lvert q_{3,1}(11)\rangle, \lvert q_{3,2}(11)\rangle)$	$\lvert q_{3,0}(11)\rangle = M_1\lvert q_{2,0}\rangle,\ \lvert q_{3,1}(11)\rangle = M_1\lvert q_{2,1}\rangle$
$s_4(x) = (\lvert q_{4,0}(x)\rangle, \lvert q_{4,1}(x)\rangle, \lvert q_{4,2}(x)\rangle)$ $(x \in \{00, 01, 10, 11\})$	$\lvert q_{4,1}(x)\rangle \otimes \lvert q_{4,2}(x)\rangle = U_{CX}\lvert q_{3,1}(x)\rangle \otimes \lvert q_{3,2}(x)\rangle$
$s_5(x) = (\lvert q_{5,0}(x)\rangle, \lvert q_{5,1}(x)\rangle, \lvert q_{5,2}(x)\rangle)$	$\lvert q_{5,0}(x)\rangle \otimes \lvert q_{5,2}(x)\rangle = U_{CZ}\lvert q_{4,0}(x)\rangle \otimes \lvert q_{4,2}(x)\rangle$
Initial valuation	$\lvert q_{0,0}\rangle \in \mathcal{H}_2,\ \lvert q_{0,1}\rangle \otimes \lvert q_{0,2}\rangle \in \{\lvert\phi^+\rangle\}$

4 The symQV Verification Algorithm

Our symQV algorithm takes as input a quantum program model $M_\mathcal{Q}$ defined in Sect. 3 and a formal specification in the form of a first-order formula φ. From that, symQV generates an SMT encoding (which we also write $M_\mathcal{Q}$ with a slight abuse of notation) as described in the previous section. Finally, this encoding together with the negated specification is asserted in a query to an SMT solver.

Theorem 2 (Soundness and completeness of the encoding). *Given a quantum program model with encoding $M_\mathcal{Q}$ and a specification φ, we have that the program satisfies φ if and only if $M_\mathcal{Q} \wedge \neg\varphi$ is unsatisfiable.*

Proof. This follows from the one-to-one correspondence of the quantum program model $M_\mathcal{Q}$ and the standard quantum circuit model [31] shown in Theorem 1. The formula is satisfiable if and only if there is an execution that violates the specification.

The formula $M_\mathcal{Q}$ falls into the theory of nonlinear real arithmetic with trigonometric expressions, for which checking satisfiability is undecidable [34]. Yet, the δ-relaxation of this problem is decidable [16]. That is why we use the δ-satisfiability framework from [17], which is implemented in dReal[2]. If the combined formula $M_\mathcal{Q} \wedge \neg\varphi$ is found to be δ-SAT, either it is indeed satisfiable (i.e., a counterexample has been found), or it is unsatisfiable (i.e., the program complies with the specification) but a δ-perturbation on its numerical terms would satisfy the formula. The parameter δ is user-controllable, and we show in the evaluation that the δ-SAT case for correct programs does not occur in practice for reasonable values of δ.

While the δ-relaxation must sacrifice completeness, it preserves soundness: If the formula is found to be unsatisfiable (UNSAT), then the quantum program is indeed correct with respect to φ.

Theorem 3 (Soundness preservation). *Let $M_\mathcal{Q}$ be the encoding of a quantum program model and φ be a specification. Assume that a δ-satisfiability solver returns* UNSAT *for the formula $M_\mathcal{Q} \wedge \neg\varphi$. Then the quantum program is correct.*

[2] Available at https://github.com/dreal/dreal4.

Proof. This follows from Theorem 2 and [17].

4.1 Running Example: Verification of Teleportation

Coming up with the right specifications for quantum programs is not trivial. Conveniently, as symQV maps all building blocks of quantum programs into an SMT representation, we have access to the full set of logic operators.

We want our specification to express that teleportation has been successful, i.e., qubit $|q_0\rangle$ has moved to where qubit $|q_2\rangle$ was at the beginning (compare the right-hand side of Fig. 3).

$$(|q_{5,2}\rangle = |q_{0,0}\rangle)$$

This, however, is not the full specification. We need to disallow operations crossing the line between the first two qubits and the last one, which only becomes possible after measurement, where the classical communication channel can be used (cf. Sect. 2.3). Therefore, we add an additional constraint that forbids state operations where these qubits appear together:

$$\varphi = (|q_{5,2}\rangle = |q_{0,0}\rangle) \land \neg\exists 0 \leq i \leq 2\colon \rightarrow_i (|q_{i,0}\rangle, |q_{i,2}\rangle) \lor \rightarrow_i (|q_{i,1}\rangle, |q_{i,2}\rangle)$$

Performing the verification is "push-button," i.e., only requires writing the quantum program model and the specification. The corresponding Python code given in Sect. A.3 of the supplementary material [4] demonstrates that a user does not have to provide any proof steps as in previous works based on proof assistants.

4.2 The symQV Over-Approximation

Encoding 1 puts trigonometric functions into the SMT formula, which are computationally expensive. This can also be later seen in the evaluation. Therefore, we introduce an over-approximation of the Hilbert space to make the verification task more efficient. This is achieved via relaxing the qubit's degrees of freedom from the unit sphere to the unit box, visualized in Fig. 4.

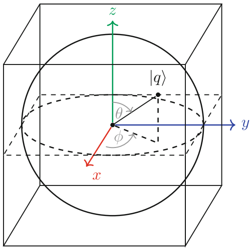


Fig. 4. The over-approximation visualized for a single qubit.

Encoding 8 (Over-approximation). *We remove the constraints in Eq.* (2) *from Encoding 1 and add the following constraint over the qubit's degrees of freedom:*

$$-1 \leq \alpha \leq 1 \quad\land\quad -1 \leq \beta_R \leq 1 \quad\land\quad -1 \leq \beta_I \leq 1. \tag{4}$$

Table 1. Benchmark quantum programs for evaluating our verification procedure. "Input" describes the input space to the quantum programs and "Parametrized" expresses whether there are parametrized gates in the quantum program.

Program	Description	Depth	Input	Parametrized
Toffoli	Toffoli Gate	5	Bit vector	No
TP	Quantum Teleportation Circuit	6	Infinite	No
ADD-8	8-bit Quantum Adder	48	Bit vector	No
QFT-n	n-Qubit Quantum Fourier Transform	$\mathcal{O}(n^2)$	Bit vector	No
QPE-n	n-Bit Quantum Phase Estimation	$\mathcal{O}(n^2)$	Concrete	Yes
GDO-n	n-Qubit Grover Diffusion Operator	$\mathcal{O}(n)$	Infinite	No

5 Evaluation

This section presents our experimental evaluation, demonstrating symQV's effectiveness in verifying several (correct) quantum programs that have symbolic inputs or symbolically parametrized quantum gates.

5.1 Implementation

symQV[3] is implemented as a Python library interfacing with dReal [17] using about 5000 lines of code. The symQV Python API allows users to specify the quantum program using a syntax inspired by Cirq [11]. The specification can be written using one of two formats:

- *State vector:* One can specify assertions on any of the 2^n vector entries.
- *Qubits:* One can specify assertions on any of the n qubits.

The logic assertions use an SMT-LIB2-compatible Python API and support specifications expressing relationships between program inputs and outputs as well as intermediate states.

5.2 Benchmark Problems and Setup

An overview of the benchmark problems is given in Table 1. Further descriptions, including the specifications, are given in Sect. A.3 of the supplementary material [4].

We compare our tool ("symQV") against quantum simulation ("Simulation"), basic SMT solving based on linear algebra ("Basic SMT"), and symQV without over-approximation ("symQV (exact)").

- *Simulation* is implemented in Qiskit [1]. The technique enumerates all possible inputs to the quantum program and then compares the outputs with the

[3] Available for download at https://doi.org/10.5281/zenodo.7400321.

specification. We can only use this technique for a finite input space, i.e., for concrete and bit-vector inputs, but neither for symbolic qubits with the entire Hilbert space $\mathcal{H}_2$ as input space, nor for parametrized gates.
- *Basic SMT* is basic SMT solving using vectors and matrices, but not using direct mappings (Definition 1).
- symQV (exact) is a modification of symQV where all over-approximation capabilities are removed, ending up with a technique that performs exact modeling, even when unnecessary (see Sect. 4.2).

We do not compare against the proof-assistant approaches [9,20,29,30,33] (cf. Sect. 1) because a comparison of run-times between an automated method, as implemented in symQV, and a semi-automated method relying on manual input is not meaningful. We also do not compare against [3] because it neither supports the full gate set nor formal logic specifications.

The experiments use the value $\delta = 10^{-4}$. We also compare the run-time of symQV for different precision levels δ.

All experiments are carried out on a workstation with an AMD Ryzen ThreadRipper 3960X @ 3.8 GHz × 24 cores processor and 256 GB RAM. The machine runs Ubuntu 20.04.3 LTS and each result is the average of 10 runs.

5.3 Results

We summarize our results in Table 2. symQV (exact) is best for quantum programs with concrete inputs or a small qubit count (TP and ADD-8); the over-approximation of symQV yields no speed-up for these instances. *Simulation* performs best for verifying combinatorial problems, i.e., for the quantum Fourier transform (QFT). Here, it can still feasibly enumerate a 12-qubit state space. Interestingly, Basic SMT scales best among the SMT-based procedures here; this is explained by the high amount of controlled operations, for which the mapping-based approach of symQV is inferior.

symQV offers a dramatic performance increase for quantum programs with symbolic inputs, i.e., quantum phase estimation (QPE) and Grover's diffusion operator (GDO). This highlights the advantage of over-approximation for this family of quantum programs. Recall that simulation is not possible for both QPE and GDO, as that would require enumerating infinitely many inputs.

The precision value $\delta = 10^{-4}$ was sufficient for all benchmarks in our evaluation. To investigate scalability in this parameter, Table 3 compares the run-times for different values for GDO with 12, 15, and 18 qubits, respectively. For the higher qubit counts, the run-time increases significantly when we lower δ to 10^{-6}, but then remains relatively stable when further tightening precision.

Overall, symQV is the strongest for quantum programs with infinite input space, i.e., programs where the (symbolic) input qubits can span the complete Hilbert space. Likewise, for programs that use parametrized quantum gates dependent on a symbolic parameter, symQV is the most effective.

Table 2. Runtime comparison results for the benchmark problems described in Table 1. "Simulation" stands for simulation and enumeration of all cases. "Basic SMT" is SMT solving with full state and matrix construction. "symQV (exact)" is symQV where over-approximations have been removed. "symQV" (this work) utilizes a sound over-approximation. "N/A" instances cannot be solved by simulation due to infinite state space. "out of memory" cases exceeded the available memory, and "timeout" cases exceed the 12-h time limit.

Benchmark	Simulation	Basic SMT	symQV (exact)	symQV
Toffoli	**0.02 s**	11.1 s	1.3 s	0.4 s
TP	N/A	44.8 s	**21.6 s**	31.0 s
ADD-8	6.1 h	out of memory	**7.6 s**	7.8 s
QFT-3	**0.005 s**	12.8 s	5.8 s	1.0 s
QFT-5	**0.03 s**	17.6 min	2.6 min	26.4 s
QFT-10	**1.5 s**	1.2 h	10.9 h	1.6 h
QFT-12	**14.0 s**	4.0 h	timeout	7.4 h
QPE-3	N/A	19.2 s	34.0 s	**8.7 s**
QPE-5	N/A	18.2 min	42.3 min	**3.9 min**
GDO-5	N/A	timeout	9.2 s	**1.3 s**
GDO-10	N/A	timeout	3.2 min	**17.0 s**
GDO-12	N/A	timeout	14.2 min	**20.2 s**
GDO-15	N/A	timeout	2.9 h	**1.0 min**
GDO-18	N/A	timeout	timeout	**4.9 min**
GDO-20	N/A	timeout	timeout	**17.1 min**
GDO-22	N/A	timeout	timeout	**1.1 h**
GDO-24	N/A	timeout	timeout	**4.2 h**

6 Discussion

Symbolic execution and formal verification scale exponentially for the quantum case, as is the case for classical software. That is to be expected: firstly, the simulation of quantum programs on classical hardware already takes exponential time and space due to the matrix representation of quantum mechanics, and secondly because the state space grows with every input variable added to the program. Nonetheless, we have shown how to keep this exponential blow-up under control by introducing mappings and over-approximations. In our evaluation, we symbolically executed quantum programs with up to 24 qubits. In comparison, even (concrete) quantum simulation for concrete inputs stops being feasible at around 30 qubits, requiring petabytes of main memory. In conclusion, symQV is most effective for unknown inputs to the quantum programs or unknown parameters of quantum gates that therefore cannot be tested.

Table 3. symQV run-time results for different precision values δ.

Delta	GDO-12	GDO-15	GDO-18
10^{-4}	20.2 s	1.0 min	4.9 min
10^{-6}	20.5 s	28.0 min	33.1 min
10^{-8}	20.8 s	49.4 min	58.7 min
10^{-10}	21.1 s	52.3 min	1.2 h

7 Conclusion

We introduced symQV, a symbolic verification technique that leverages overapproximation to make automated verification of quantum programs feasible. We formalized quantum program semantics in SMT and proposed a sound overapproximation that allows scaling to realistic program sizes. Thanks to the symbolic nature of our approach, we can analyze quantum programs with infinite input space, which is beyond the capabilities of quantum simulation. We demonstrate these achievements by formally verifying multiple quantum programs against their specifications within a modest time frame.

In this paper, we focused on formalizing the mathematical foundations to model quantum programs, define specifications, and prove their specification compliance. We intend this to be the first step in a larger, fully automated quantum verification framework, including counterexample-guided refinement. In the future, we will investigate strategies that allow us to verify hybrid programs that perform classical and quantum computations.

Acknowledgments. This research was partly supported by DIREC - Digital Research Centre Denmark and the Villum Investigator Grant S4OS.

References

1. Abraham, F.N., et al.: Qiskit: an open-source framework for quantum computing (2017). https://github.com/Qiskit
2. Ajagekar, A., Humble, T., You, F.: Quantum computing based hybrid solution strategies for large-scale discrete-continuous optimization problems. Comput. Chem. Eng. **132** (2020). https://doi.org/10.1016/j.compchemeng.2019.106630
3. Amy, M.: Towards large-scale functional verification of universal quantum circuits. In: QPL. EPTCS, vol. 287, pp. 1–21 (2018). https://doi.org/10.4204/EPTCS.287.1
4. Bauer-Marquart, F., Leue, S., Schilling, C.: symQV: automated symbolic verification of quantum programs. CoRR, abs/2212.02267 (2022). https://doi.org/10.48550/arXiv.2212.02267
5. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions. TTCS. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-662-07964-5
6. Beyer, D., Keremoglu, M.E.: CPAchecker: a tool for configurable software verification. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp.

184–190. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_16

7. Cadar, C., Dunbar, D., Engler, D.R.: KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In: OSDI, vol. 8, pp. 209–224. USENIX Association (2008). http://www.usenix.org/events/osdi08/tech/full_papers/cadar/cadar.pdf
8. Centrone, F., Kumar, N., Diamanti, E., Kerenidis, I.: Experimental demonstration of quantum advantage for NP verification with limited information. Nat. Commun. **12**(1), 850 (2021). https://doi.org/10.1038/s41467-021-21119-1
9. Chareton, C., Bardin, S., Bobot, F., Perrelle, V., Valiron, B.: An automated deductive verification framework for circuit-building quantum programs. In: Yoshida, N. (ed.) ESOP 2021. LNCS, vol. 12648, pp. 148–177. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72019-3_6
10. Childs, A.M., Maslov, D., Nam, Y.S., Ross, N.J., Su, Y.: Toward the first quantum simulation with quantum speedup. Proc. Natl. Acad. Sci. U.S.A. **115**(38), 9456–9461 (2018). https://doi.org/10.1073/pnas.1801723115
11. Cirq Developers. Cirq (2021). See full list of authors on Github: https://github.com/quantumlib/Cirq/graphs/contributors
12. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.): Handbook of Model Checking. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-10575-8
13. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL, pp. 238–252. ACM (1977). https://doi.org/10.1145/512950.512973
14. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
15. Farhi, E., Goldstone, J., Gutmann, S.: A quantum approximate optimization algorithm. arXiv preprint (2014). https://doi.org/10.48550/arXiv.1411.4028
16. Gao, S., Avigad, J., Clarke, E.M.: δ-complete decision procedures for satisfiability over the reals. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 286–300. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31365-3_23
17. Gao, S., Kong, S., Clarke, E.M.: dReal: an SMT solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) CADE 2013. LNCS (LNAI), vol. 7898, pp. 208–214. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38574-2_14
18. Goddard, P., Mniszewski, S., Neukart, F., Pakin, S., Reinhardt, S.: How will early quantum computing benefit computational methods? In: Proceedings of the SIAM Annual Meeting (2017). https://sinews.siam.org/Details-Page/how-will-early-quantum-computing-benefit-computational-methods
19. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: STOC, pp. 212–219. ACM (1996). https://doi.org/10.1145/237814.237866
20. Hietala, K., Rand, R., Hung, S., Li, L., Hicks, M.: Proving quantum programs correct. In: ITP, Dagstuhl, Germany. LIPIcs, vol. 193, pp. 21:1–21:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.ITP.2021.21
21. Houssein, E.H., Abohashima, Z., Elhoseny, M., Mohamed, W.M.: Hybrid quantum convolutional neural networks model for COVID-19 prediction using chest X-ray images. CoRR (2021). https://arxiv.org/abs/2102.06535
22. IBM. IBM's roadmap for scaling quantum technology (2020). https://research.ibm.com/blog/ibm-quantum-roadmap

23. Jerbi, S., Fiderer, L.J., Nautrup, H.P., Kübler, J.M., Briegel, H.J., Dunjko, V.: Quantum machine learning beyond kernel methods. CoRR (2021). https://arxiv.org/abs/2110.13162
24. Jordan, S.: Quantum algorithm zoo (2021). https://quantumalgorithmzoo.org
25. Kadowaki, T., Nishimori, H.: Quantum annealing in the transverse Ising model. Phys. Rev. E **58**(5) (1998). https://doi.org/10.1103/PhysRevE.58.5355
26. Krinner, S., et al.: Realizing repeated quantum error correction in a distance-three surface code. Nature **605**(7911), 669–674 (2022). https://doi.org/10.1038/s41586-022-04566-8
27. Kroening, D., Tautschnig, M.: CBMC – C bounded model checker. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 389–391. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_26
28. Li, G., Zhou, L., Yu, N., Ding, Y., Ying, M., Xie, Y.: Projection-based runtime assertions for testing and debugging quantum programs. Proc. ACM Program. Lang. **4**(OOPSLA), 150:1–150:29 (2020). https://doi.org/10.1145/3428218
29. Liu, J., et al.: Formal verification of quantum algorithms using quantum Hoare logic. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11562, pp. 187–207. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25543-5_12
30. Liu, S., et al.: $Q|SI\rangle$: a quantum programming environment. In: Jones, C., Wang, J., Zhan, N. (eds.) Symposium on Real-Time and Hybrid Systems. LNCS, vol. 11180, pp. 133–164. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01461-2_8
31. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information (10th Anniversary edition). Cambridge University Press (2016). https://doi.org/10.1017/CBO9780511976667. ISBN 978-1-10-700217-3
32. Nipkow, T., Wenzel, M., Paulson, L.C. (eds.): Isabelle/HOL - A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45949-9
33. Rand, R., Paykin, J., Zdancewic, S.: QWIRE practice: formal verification of quantum circuits in Coq. In: QPL. EPTCS, vol. 266, pp. 119–132 (2017). https://doi.org/10.4204/EPTCS.266.8
34. Richardson, D.: Some undecidable problems involving elementary functions of a real variable. J. Symb. Log. **33**(4), 514–520 (1968). https://doi.org/10.2307/2271358
35. Shi, Y., et al.: CertiQ: a mostly-automated verification of a realistic quantum compiler. arXiv preprint (2019). https://doi.org/10.48550/arXiv.1908.08963
36. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997). https://doi.org/10.1137/S0097539795293172
37. Svore, K.M., et al.: Q#: enabling scalable quantum computing and development with a high-level DSL. In: RWDSL, pp. 7:1–7:10. ACM (2018). https://doi.org/10.1145/3183895.3183901
38. Traversa, F.L.: Aircraft loading optimization: MemComputing the 5th Airbus problem. CoRR, abs/1903.08189 (2019). http://arxiv.org/abs/1903.08189
39. Yu, N., Palsberg, J.: Quantum abstract interpretation. In: PLDI, pp. 542–558. ACM (2021). https://doi.org/10.1145/3453483.3454061