AutoQ: An Automata-based Quantum Circuit Verifier

Yu-Fang Chen[1], Kai-Min Chung[1], Ondřej Lengál[2], Jyun-Ao Lin[1], and Wei-Lun Tsai[1]
yfc@iis.sinica.edu.tw, lengal@fit.vutbr.cz, alan23273850@gmail.com

[1] Institute of Information Science, Academia Sinica, Taipei, Taiwan
[2] Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic

Abstract. We present a specification language and a fully automated tool named AutoQ for verifying quantum circuits symbolically. The tool implements the automata-based algorithm from [14] and extends it with the capabilities for symbolic reasoning. The extension allows to specify *relational* properties, i.e., relationships between states before and after executing a circuit. We present a number of use cases where we used AutoQ to fully automatically verify crucial properties of several quantum circuits, which have, to the best of our knowledge, so far been proved only with human help.

1 Introduction

Recently, quantum computing has received much attention, driven by several technological breakthroughs [7] and increasing investments. Prototype quantum computers are already available. The opportunities for the general public—particularly students, researchers, and technology enthusiasts—to access quantum computing devices are rapidly increasing, e.g., through cloud services such as Amazon Braket [1] or IBM Quantum [2]. Due to the complexity and probabilistic nature of quantum computing, the chance of errors in quantum programs is much higher than that of traditional programs, and conventional means for correctness assurance, such as testing, are much less applicable in the quantum world. Quantum programmers need better tools to help them write correct programs. Therefore, researchers anticipate that formal verification will play a crucial role in quantum software quality assurance and have, in recent years, invested significant effort in this direction [21,41,42,11,5,45,46,43]. Nevertheless, practical tools for automated quantum program/circuit verification are still missing.

This paper introduces AutoQ[3], a fully automated tool for quantum circuit verification based on the approach proposed in [14]. In particular, AutoQ checks the validity of a Hoare-style specification $\{\texttt{Pre}\}\ C\ \{\texttt{Post}\}$, where C is a quantum circuit (a sequence of quantum gates) in the OpenQASM format [17] and the precondition `Pre` and postcondition `Post` represent sets of (pure) quantum states. The check is done by executing the circuit with all quantum states satisfying `Pre` (using a symbolic representation) and testing that all resulting quantum states are in the set denoted by `Post`.

AutoQ combines two main techniques to efficiently and effectively represent and reason about (potentially infinite) sets of quantum states:

[3] Available at https://github.com/alan23273850/AutoQ

1. As in [14], we use *tree automata* (TAs), finite-state automata accepting languages of trees, to efficiently represent *sets* of quantum states: Each quantum state over n qubits can be seen as a binary decision tree over n variables such that, e.g., in a 3-qubit circuit with qubits $|x_1x_2x_3\rangle$, if the computational basis state $|010\rangle$ in a quantum state has the probability amplitude $\frac{1}{4}$, then there will be a branch $x_1 \xrightarrow{0} x_2 \xrightarrow{1} x_3 \xrightarrow{0} \frac{1}{4}$ in the corresponding tree. The use of TA-based representation of a set of quantum states has several advantages: (a) It is *concise*: e.g., in order to represent the set of all 2^n basis states of an n-qubit quantum circuit, we suffice with a TA with $\mathcal{O}(n)$ states and transitions. (b) It allows to *efficiently perform quantum gate operations* on the whole set of quantum states represented by a TA at once [14].
2. In this work, we further consider *symbolic quantum states*, represented by assigning symbolic values to computational basis states (and having an additional formula to relate these symbolic values). For instance, we can represent the set of all n-qubit quantum states where the computational basis $|0\ldots0\rangle$ has a strictly larger probability of measurement than all other basis states by a symbolic quantum state assigning $|0\ldots0\rangle \mapsto v_h$ and $|y_1\ldots y_n\rangle \mapsto v_\ell$ for all $y_1\ldots y_n \neq 0\ldots0$, together with the formula $|v_h|^2 > |v_\ell|^2 \wedge |v_h|^2 + (2^n-1)|v_\ell|^2 = 1$, where v_h and v_ℓ are symbolic variables ranging over complex numbers

By combining these two techniques, i.e., using TAs with symbolic variables in leaves, we can have a representation of all n-qubit quantum states where an arbitrary basis has a strictly larger amplitude than other basis states using $\mathcal{O}(n)$ states and transitions.

Using such a symbolic encoding is essential to allow us to describe *relational specifications*, e.g., it allows us to express properties like "the probability amplitude of the basis state $|000\rangle$ is increased after executing the circuit C" (for this, in the postcondition, we use TAs accepting trees with *predicates* in leaves, a subclass of symbolic tree automata of [36]). Such a property can then be verified by executing the quantum circuit *symbolically* in the spirit of symbolic execution [27] (i.e., such that the values of amplitudes are not complex numbers but, instead, *symbolic terms*) and checking whether all trees in the language of the resulting TA satisfy the desired property (using a modified antichain-based algorithm for testing TA language inclusion [10,4]). Combining TAs and symbolic variables as the language for quantum predicates allows full automation and can be used to express many crucial properties of quantum circuits, as we will demonstrate later. AutoQ is the first tool implementing this approach.

Related work. Our work belongs to the line of *Hoare-style verification* of quantum programs, which has been widely discussed in the past [44,35,22,40,29]. This family of approaches follows D'Hondt and Panangaden's suggestion of using various Hermitian operators as quantum predicates, resulting in a very powerful yet complete proof system [20]. However, specifying properties using Hermitian operators is often not intuitive and is inconvenient for automation due to their enormous matrix sizes. Therefore, often these approaches are implemented on top of proof assistants such as Coq [9] and Isabelle [37] and require significant manual work in proof search. The Qbricks [12] approach alleviates the difficulty of the proof search by combining state-of-the-art theorem provers with decision procedures building on top of the Why3 platform [24]. The approach, however, still requires a significant amount of human intervention.

Regarding other quantum program/circuit/protocol verification tools, *circuit equivalence checkers* [5,15,26,39,11] are often quite efficient but less flexible in specifying the desired property (only equivalence). They are particularly useful in *compiler validation*; notable tools include QCEC [11], and FEYNMAN [5]. *Quantum model checking* supports a rich specification language (flavors of temporal logic [23,30,38]) and is more suitable for *verifying high-level protocols* due to the quite limited scalability [6]. One notable tool in this category is QPMC [23]. *Quantum abstract interpretation* [43,32] is particularly efficient in processing large-scale circuits, but it grossly over-approximates the state space (it cannot verify basic properties of, e.g., Grover's algorithm) and cannot conclude anything when verification fails. In contrast, AUTOQ can be conveniently used for quantum program development and debugging since it automatically computes the exact set of reachable states[4]. The mentioned tools are fully automated but have different goals or address different parts of the software development cycle than AUTOQ.

Contributions. AUTOQ evolved from a simple prototype used for performance evaluation in [14] into a robust tool. In addition, we added the following major extensions:

1. We combined the TA specification with symbolic variables, allowing users to specify advanced relational properties of quantum circuits.
2. We developed a new entailment-checking algorithm for the symbolic TA specification based on the antichain algorithm for automata language inclusion testing.
3. We introduced a high-level language to simplify writing TA specifications.

These improvements are pushing the capabilities of AUTOQ, and also of practical quantum circuit verification itself, much further.

Outline. In Section 2, we describe our approach to TA-based specification and verification of quantum circuits. In Section 3, we discuss the new entailment-checking algorithm for the symbolic TA representation. We discuss the architecture of AUTOQ in Section 4 and demonstrate the use of the specification language and AUTOQ for automated verification of several case studies in Section 5.

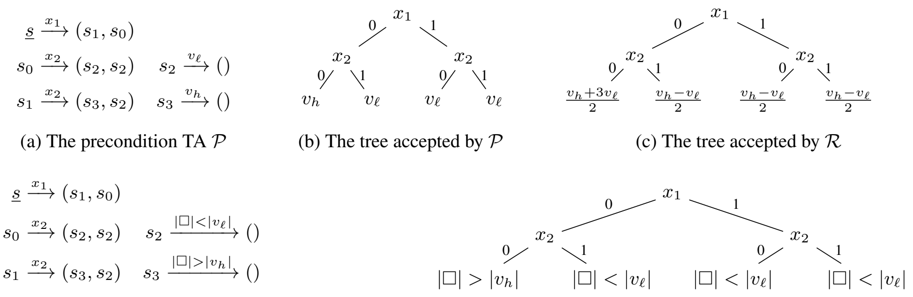
2 Tree Automata-based Verification of Quantum Circuits

We will begin with minimal formal definitions of the TA-based specification and demonstrate how to use them to verify quantum circuits in AUTOQ with examples. We assume a basic knowledge of quantum computation (see, e.g., the classical textbook [31]).

Let us fix a finite set of *quantum variables* $\mathbb{X} = \{x_1, \ldots, x_n\}$ with a linear ordering (we assume $x_1 < \ldots < x_n$) and a disjoint non-empty leaf alphabet Σ. We will, in particular, work with $\Sigma = \Sigma_t \uplus \Sigma_p$ where Σ_t is the alphabet of *terms* and Σ_p is the alphabet of *predicates* in a suitable first-order theory (discussed later).

We use $\{0,1\}^{\leq n}$ to denote $\bigcup_{0 \leq i \leq n} \{0,1\}^i$. A *(symbolic binary decision) tree* over $\mathbb{X}$ and Σ is a function $\tau\colon \{0,1\}^{\leq n} \to (\mathbb{X} \cup \Sigma)$ such that for all positions $p \in \{0,1\}^i$ with $i < n$, we have $\tau(p) = x_{i+1}$ and for all positions $p \in \{0,1\}^n$, we have $\tau(p) \in \Sigma$. An example of a tree τ can be found in Fig. 1b, where $\Sigma = \{v_h, v_\ell\}$, $\tau(\epsilon) = x_1$, $\tau(0) = \tau(1) = x_2$, $\tau(00) = v_h$, and $\tau(p) = v_\ell$ for $p \in \{0,1\}^2 \setminus \{00\}$.

[4] A predecessor of the presented version of AUTOQ has already caught a bug in QCEC, cf. [3].



$\underline{s} \xrightarrow{x_1} (s_1, s_0)$
$s_0 \xrightarrow{x_2} (s_2, s_2)$ $s_2 \xrightarrow{v_\ell} ()$
$s_1 \xrightarrow{x_2} (s_3, s_2)$ $s_3 \xrightarrow{v_h} ()$
x_1 0 1 x_2 0 1 x_2 0 1
v_h v_ℓ v_ℓ v_ℓ
$\frac{v_h+3v_\ell}{2}$ $\frac{v_h-v_\ell}{2}$ $\frac{v_h-v_\ell}{2}$ $\frac{v_h-v_\ell}{2}$
$\underline{s} \xrightarrow{x_1} (s_1, s_0)$
$s_0 \xrightarrow{x_2} (s_2, s_2)$ $s_2 \xrightarrow{|\square|<|v_\ell|} ()$
$s_1 \xrightarrow{x_2} (s_3, s_2)$ $s_3 \xrightarrow{|\square|>|v_h|} ()$
$|\square| > |v_h|$ $|\square| < |v_\ell|$ $|\square| < |v_\ell|$ $|\square| < |v_\ell|$

(a) The precondition TA $\mathcal{P}$

(b) The tree accepted by $\mathcal{P}$

(c) The tree accepted by $\mathcal{R}$

(d) The postcondition TA $\mathcal{Q}$

(e) The tree accepted by $\mathcal{Q}$

Fig. 1: Verification of a circuit C amplifying the amplitude of $|00\rangle$ w.r.t. the specification $\{\mathcal{P}, \varphi\}\ C\ \{\mathcal{Q}\}$ with $\varphi\colon |v_h + 3v_\ell| > |2v_h|$. $\mathcal{R}$ is the TA obtained by executing $\mathcal{P}$ on C.

A *(symbolic) tree automaton* (TA) is a tuple $\mathcal{A} = (S, \Delta, F)$ where S is a finite set of *states*, $\Delta \subseteq (S \times \mathbb{X} \times S \times S) \cup (S \times \Sigma)$ is a *transition relation*, and $F \subseteq S$ is the set of *root (final) states*. We denote transitions from Δ as $s \xrightarrow{x_i} (s_0, s_1)$ and $s \xrightarrow{a} ()$ respectively. An example of a TA with the set of root states $\{s\}$ can be found in Fig. 1a.

A *run* of $\mathcal{A}$ on τ is a function $\rho\colon \{0,1\}^{\leq n} \to S$ s.t. for all positions $p \in \{0,1\}^i$ with $i < n$, it holds that $\rho(p) \xrightarrow{\tau(p)} (\rho(p.0), \rho(p.1)) \in \Delta$ and for all positions $p \in \{0,1\}^n$, it holds that $\rho(p) \xrightarrow{\tau(p)} () \in \Delta$. The run ρ is *accepting* iff $\rho(\epsilon) \in F$ and the *language* of $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \{\tau \mid \mathcal{A}$ has an accepting run on $\tau\}$. Observe that the tree in Fig. 1b is in the language of the TA $\mathcal{P}$ in Fig. 1a with the run ρ such that $\rho(\epsilon) = s$, $\rho(0) = s_1$, $\rho(1) = s_0$, $\rho(00) = s_3$, and $\rho(p) = s_2$ for $p \in \{0,1\}^2 \setminus \{00\}$.

Now we are ready to demonstrate how to write specifications of quantum circuits with TAs using a running example. We assume that C is a 2-qubit circuit that amplifies the amplitude of the basis state $|00\rangle$ (under some constraint φ over input states) and reduces the amplitudes of other basis states. We first prepare the precondition of C, which consists of a pair $(\mathcal{P}, \varphi)$, where $\mathcal{P}$ is a TA with the root state s, a set of terms Σ_t as the leaf alphabet, and the set of transitions from Fig. 1a, and φ is a first-order constraint over the variables used in Σ_t. In Σ_t, we use two variables over complex numbers, v_ℓ and v_h, to denote the corresponding amplitude (*low* and *high*). The constraint φ states that $|v_h + 3v_\ell| > |2v_h|$ (required by this circuit C, cf. Section 5.4). Recall that the TA $\mathcal{P}$ from Fig. 1a accepts the tree from Fig. 1b, which in turn represents the quantum state

$$s = v_h\,|00\rangle + v_\ell\,|01\rangle + v_\ell\,|10\rangle + v_\ell\,|11\rangle\,. \tag{1}$$

AutoQ will execute the gates in C to transform the TA $\mathcal{P}$ to another TA $\mathcal{R}$ capturing the effect of executing C over all quantum states encoded in $\mathcal{P}$. The algorithm for gate operations is almost the same as the one in [14], except that now the update of leaf symbols works symbolically (similarly to symbolic execution [27]: each leaf symbol is a term over v_h and v_ℓ and quantum gates change the terms by accumulating the operations that would be performed on them, potentially simplifying them). In this example, the TA $\mathcal{R}$ will accept only one tree representing the quantum state

$$s' = (\tfrac{v_h+3v_\ell}{2})\,|00\rangle + (\tfrac{v_h-v_\ell}{2})\,|01\rangle + (\tfrac{v_h-v_\ell}{2})\,|10\rangle + (\tfrac{v_h-v_\ell}{2})\,|11\rangle\,, \tag{2}$$

Observe that under the precondition $\varphi = |v_h + 3v_\ell| > |2v_h|$, the probability of $|00\rangle$ is indeed increased ($|\frac{v_h+3v_\ell}{2}|^2 > |v_h|^2$). The tree representation of s' can be found in Fig. 1c. The TA $\mathcal{Q}$ of the postcondition can be found in Fig. 1d. The leaf alphabet of $\mathcal{Q}$ is the set of predicates $\Sigma_p = \{|\Box| > |v_h|, |\Box| < |v_\ell|\}$ where $\Box$ denotes a free variable. Observe that $\mathcal{Q}$ accepts the tree from Fig. 1e.

2.1 High-Level Specification Language

In AutoQ, we provide a simple specification language that can be automatically translated to TAs. The language allows users to focus on the properties they want to express without the need to specify details of the TA structure. Our language is particularly suitable for describing sets of states with one high probability branch and other branches with uniformly low or zero probability, a very common pattern of quantum circuit's correctness properties. For example, in the language, we can use $(|00\rangle\colon v_h, |{*}\rangle\colon v_\ell)$, where "$|{*}\rangle$" denotes "other basis states," to define the tree language of the TA in Fig. 1a, which accepts a single tree representing the quantum state $v_h |00\rangle + v_\ell |01\rangle + v_\ell |10\rangle + v_\ell |11\rangle$ from Fig. 1b. Similarly, we can use $(|00\rangle\colon |\Box| > |v_h|, |{*}\rangle\colon |\Box| < |v_\ell|)$ to represent the language of the TA in Fig. 1d. The set of all 2-qubit basis states $\{|i\rangle \mid i \in \{0,1\}^2\}$ is expressed as $\exists i \in \{0,1\}^2\colon (|i\rangle\colon 1, |{*}\rangle\colon 0)$ (we can see it as a predicate that is satisfied by the described quantum states). We also allow the *tensor product* $\otimes$ operator, which multiplies the amplitude of the product basis states. For example, $(|00\rangle\colon 1, |{*}\rangle\colon 0) \otimes (|00\rangle\colon v_h, |{*}\rangle\colon v_\ell) \otimes (|00\rangle\colon 1, |{*}\rangle\colon 0)$ represents the (singleton) set of states compactly $\{v_h |000000\rangle + \sum_{j\in\{01,11,10\}} v_\ell |00j00\rangle\}$.

A more challenging example is to represent the set of states

$$\left\{ v_h |ii000\rangle + \sum_{j\in\{0,1\}^3 \wedge j\neq i} v_\ell |ij000\rangle \;\middle|\; i \in \{0,1\}^3 \right\}. \tag{3}$$

Such a set can be described with the help of the $\otimes$ and $\exists$ operators as follows:

$$\exists i \in \{0,1\}^3\colon (|i\rangle\colon 1, |{*}\rangle\colon 0) \otimes (|i\rangle\colon v_h, |{*}\rangle\colon v_\ell) \otimes (|000\rangle\colon 1, |{*}\rangle\colon 0). \tag{4}$$

Below is the grammar of specification *spec*:

$$\begin{aligned} \mathit{spec} &::= \mathit{state} \;\big|\; \exists i \in \{0,1\}^n\colon \mathit{state} \;\big|\; \mathit{spec}, \mathit{state} \\ \mathit{state} &::= (|c_1\rangle\colon t, \ldots, |c_k\rangle\colon t, |{*}\rangle\colon t) \;\big|\; (|i\rangle\colon t, |{*}\rangle\colon t) \;\big|\; \mathit{state} \otimes \mathit{state} \\ & \quad t \in \Sigma,\ n \in \mathbb{N}, \text{ and } c_1, \ldots, c_k \in \{0,1\}^n \end{aligned}$$

A *spec* is ill-formed when a free variable i appears in state, if some basis is repeated in the rule $(|c_1\rangle\colon t, \ldots, |c_k\rangle\colon t, |{*}\rangle\colon t)$, or if the previous rule contains two bases of different lengths. If all basis states of the given length are specified in $(|c_1\rangle\colon t, \ldots, |c_k\rangle\colon t, |{*}\rangle\colon t)$, the $|{*}\rangle\colon t$ part is not required any more. The specification is then converted into a TA using a straightforward algorithm; in the following we often confuse a TA and its specification.

2.2 Complex Number Representation

In a (pure) quantum state, the amplitude of a basis computational state is a *complex number*, and the corresponding probability is the square of the absolute value of the amplitude. For verification, we need an exact representation of complex numbers that can be used in computers. In AutoQ, we use a subset of complex numbers that can be expressed by the following algebraic encoding (cf. [46,34,14]):

$$\left(\frac{1}{\sqrt{2}}\right)^k (a + b\omega + c\omega^2 + d\omega^3), \tag{5}$$

where $a, b, c, d \in \mathbb{Z}$, $k \in \mathbb{N}$, and $\omega = e^{\frac{i\pi}{4}} = \cos 45^\circ + i \sin 45^\circ = \frac{\sqrt{2}}{2} + i\frac{\sqrt{2}}{2}$, the unit vector that makes an angle of 45° with the positive real axis in the complex plane. A complex number is then represented by a quadruple (a, b, c, d) of integers and a normalization factor k. Although the considered set of complex numbers is only a small subset of all complex numbers (it is countable, while the set of all complex numbers is uncountable), the subset is sufficient to describe various standard quantum gates. Currently, AutoQ supports the set of quantum gates X, H, Y, Z, S, T, Rx($\frac{\pi}{2}$), Ry($\frac{\pi}{2}$), CNOT, CZ, Toffoli (cf. the list in [14]), which already includes a set of universal quantum gates. From the Solovay-Kitaev theorem [18], gates performing rotations of $\frac{\pi}{2^n}$, used, e.g., in Shor's algorithm [33] and *quantum Fourier transform* (QFT) [16], can be approximated with an error rate ϵ by $\mathcal{O}(\log^{3.97}(\frac{1}{\epsilon}))$-many H, CNOT, and T gates. The algebraic representation is also sufficient to represent all reachable states in OpenQASM circuits with the set of supported gates, where the initial basis state is $|0 \ldots 0\rangle$.

AutoQ operates on the introduced representation of complex numbers. More precisely, for a specification $\{\mathcal{P}, \varphi\}\ C\ \{\mathcal{Q}\}$, the leaf symbols of $\mathcal{P}$ are quadruples of integer terms (a, b, c, d). We assume that all leaf symbols of $\mathcal{P}$ share a common normalization factor k, so we do not store the value of k explicitly since it can be inferred from the fact that the probability sum over all basis states is one. Instead, we remember a constant natural number value k_c, the difference of the k value between $\mathcal{P}$ and $\mathcal{R}$, and use it to normalize the amplitudes. Recall that $\mathcal{R}$ is the TA accepting all states after executing C from some states accepted by $\mathcal{P}$. The initial value of k_c is zero, and each application of H, Rx($\frac{\pi}{2}$), or Ry($\frac{\pi}{2}$) gates will increase it by one (cf. [14]). We normalize all quadruple leaf symbols (a, b, c, d) of $\mathcal{R}$ by multiplying them with $\left(\frac{1}{\sqrt{2}}\right)^{k_c}$ once $\mathcal{R}$ is computed.

Next, we show how to compose a specification of our running example from Fig. 1 using the algebraic representation. The specification can now be written as

$\mathcal{P}$: ($|00\rangle$: $(v_h^a, v_h^b, v_h^c, v_h^d)$, $|*\rangle$: $(v_\ell^a, v_\ell^b, v_\ell^c, v_\ell^d)$)
$\mathcal{Q}$: ($|00\rangle$: $|(\Box_1, \Box_2, \Box_3, \Box_4)|^2 > |(v_h^a, v_h^b, v_h^c, v_h^d)|^2$, $|*\rangle$: $|(\Box_1, \Box_2, \Box_3, \Box_4)|^2 < |(v_\ell^a, v_\ell^b, v_\ell^c, v_\ell^d)|^2$),

where
$$\begin{aligned} |(a, b, c, d)|^2 &= |a + b\omega + c\omega^2 + d\omega^3|^2 \\ &= \left|a + b(\tfrac{\sqrt{2}}{2} + \tfrac{\sqrt{2}}{2}i) + ci + d(-\tfrac{\sqrt{2}}{2} + \tfrac{\sqrt{2}}{2}i)\right|^2 \\ &= (a + b\tfrac{\sqrt{2}}{2} - d\tfrac{\sqrt{2}}{2})^2 + (b\tfrac{\sqrt{2}}{2} - c + d\tfrac{\sqrt{2}}{2})^2 \end{aligned}$$

2.3 Precise Semantics of the Specification

As mentioned above, for verifying $\{\mathcal{P}, \varphi\}\ C\ \{\mathcal{Q}\}$, we start with a TA $\mathcal{P}$ representing the set of all quantum states satisfying the precondition and compute a TA $\mathcal{R}$ representing the set of states reachable after executing the circuit C. Then, we test whether $\mathcal{R}$ entails $\mathcal{Q}$ (w.r.t. φ), i.e., whether all reachable states satisfy the postcondition.

Algorithm 1: Checking $\mathcal{R} \models_\varphi \mathcal{Q}$

```
   Input: A TA $\mathcal{R} = (S_r, \Delta_r, F_r)$, a TA $\mathcal{Q} = (S_q, \Delta_q, F_q)$, a formula $\varphi$
   Output: $true$ if $\mathcal{R} \models_\varphi \mathcal{Q}$, $false$ otherwise
1  $Processed \leftarrow \emptyset$;
2  $Worklist \leftarrow Min\{(s_r, U_q) \mid s_r \xrightarrow{t_r} () \in \Delta_r,$
3          $U_q = \{u_q \in Q_q \mid u_q \xrightarrow{t_r} () \vee \exists u_q \xrightarrow{p_q} () \in \Delta_q : \varphi \Rightarrow p_q[t_r/\Box]\}\}$;
4  while $Worklist \neq \emptyset$ do
5      $(s_r, U_q) \leftarrow Worklist.pop()$;
6      if $s_r \in F_r \wedge U_q \cap F_q = \emptyset$ then return $false$ ;
7      $Processed \leftarrow Min(Processed \cup \{(s_r, U_q)\})$;
8      $tmp \leftarrow (\{(s_r, U_q)\} \times Processed) \cup (Processed \times \{(s_r, U_q)\})$;
9      foreach $((s^1_r, U^1_q), (s^2_r, U^2_q)) \in tmp, \alpha \in \mathbb{X}$ do
10         $H_r \leftarrow \{s'_r \in Q_r \mid s'_r \xrightarrow{\alpha} (s^1_r, s^2_r) \in \Delta_r\}$;
11         $U'_q \leftarrow \{s_q \in Q_q \mid \exists s^1_q \in U^1_q, \exists s^2_q \in U^2_q : s_q \xrightarrow{\alpha} (s^1_q, s^2_q) \in \Delta_q\}$;
12         foreach $s'_r \in H_r$ s.t. $(s'_r, U'_q) \notin \lceil Processed \cup Worklist \rceil$ do
13             $Worklist \leftarrow Min(Worklist \cup \{(s'_r, U'_q)\})$;
14 return $true$;
```

Formally, we say that a tree τ_1 is *entailed* by a tree τ_2 w.r.t. a first-order formula φ, denoted as $\tau_1 \models_\varphi \tau_2$, if for all positions $p \in \{0,1\}^n$ it holds that either (i) $\tau_1(p) = \tau_2(p)$ or (ii) $\tau_1(p) = (t_1, \ldots, t_k) \in \Sigma_t$, $\tau_2(p) = \psi \in \Sigma_p$, and $\varphi \Rightarrow \psi[t_1/\Box_1] \ldots [t_k/\Box_k]$. We lift the entailment to TAs: $\mathcal{A}_1 \models_\varphi \mathcal{A}_2$ iff for all trees $\tau_1 \in \mathcal{L}(\mathcal{A}_1)$ there exists a tree $\tau_2 \in \mathcal{L}(\mathcal{A}_2)$ s.t. $\tau_1 \models_\varphi \tau_2$.[5]

3 Entailment Checking

We will now describe how we perform the entailment check $\mathcal{R} \models_\varphi \mathcal{Q}$. Since we operate with trees and tree automata over symbolic values, we cannot establish entailment by running a classical TA language inclusion test based on complementing the automaton $\mathcal{Q}$ first. Instead, our algorithm for testing the entailment $\mathcal{R} \models_\varphi \mathcal{Q}$ is based on an on-the-fly TA inclusion checking algorithm [10,4], which avoids complementation. The on-the-fly inclusion-checking algorithm can be seen as an optimization of the classical construction, which would establish $\mathcal{L}(\mathcal{R}) \cap \overline{\mathcal{L}(\mathcal{Q})} \overset{?}{=} \emptyset$ by first computing the complement $\mathcal{Q}^{\complement}$ of $\mathcal{Q}$ (using a bottom-up TA determinization), followed by computing the intersection $\mathcal{A}_\cap$ of $\mathcal{Q}^{\complement}$ and $\mathcal{R}$, and, finally, checking language emptiness of $\mathcal{A}_\cap$. In particular, the on-the-fly inclusion checking algorithm can be seen as doing all the operations at once. Furthermore, the algorithms in [10,4] also make use of the so-called *antichains* and TA *simulation* to prune the explored state space.

Our modification of the inclusion algorithm to test TA entailment, given in Algorithm 1, mainly differs from [10,4] in the way initial sets of state pairs are computed on Line 3. In particular, we match a state s_r that can perform a leaf transition over t_r in $\mathcal{R}$ with the set U_q of all states in $\mathcal{Q}$ that can perform a leaf transition either over t_r or over

[5] We never have a predicate from Σ_p on the left-hand side of the entailment test, so we do not need to test implication between predicates, which would be needed for a complete procedure.

a predicate p_q such that $\varphi \Rightarrow p_q[t_r/\Box]$ (we use $p_q[t_r/\Box]$ for a tuple t_r to denote the substitution of the tuple's components into the corresponding free variables of p_q).

After that, the algorithms perform a simultaneous bottom-up traversal through $\mathcal{R}$ (represented by states s_r) and the determinized version of $\mathcal{Q}$ (represented by sets of states U_q). For each such pair (s_r, U_q), the algorithm first checks whether s_r is a root state and U_q does not contain any root state (cf. Line 6; this would mean that $\mathcal{R}$ accepts some tree that is not accepted by $\mathcal{Q}$). If this does not hold, then the algorithm tries to find all already processed pairs that can make a transition with (s_r, U_q) (cf. Line 8) and continue from all such pairs. Each bottom-up successor (s'_r, U'_q) is then added to *Worklist* in the case it has not been seen previously (cf. Line 13).

The algorithm uses the function *Min* (cf. Lines 3, 7, and 13) to minimize the sets *Worklist* and *Processed* w.r.t. a subsumption relation, and the downward closure for $\lceil \mathit{Processed} \cup \mathit{Worklist} \rceil$ on Line 12 to prune the explored state space. Due to lack of space, we refer to the works [10,4] for more details about these optimizations.

4 Architecture

We illustrate the architecture of AutoQ in Fig. 2. The tool is written in C++ and uses the following external tools: the TA library Vata [28] for efficient testing of TA inclusion (when the postcondition uses only the term alphabet Σ_t) and the SMT solver Z3 for entailment checking of leaf symbols in Algorithm 1. We allow any theory solver supported by Z3. In our experiment, we use `QF_NIRA`. AutoQ takes as an input a quantum circuit in the OpenQASM format accompanied with the specification written as tree automata (`.aut` files) or the high-level specification language (`.hsl` files) introduced in Section 2.1.

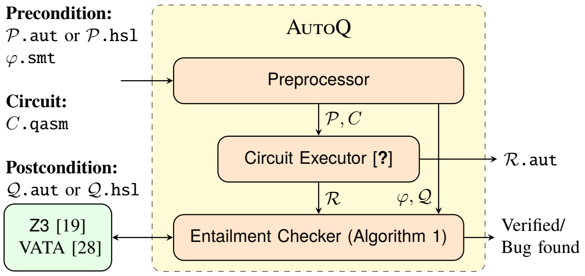


Fig. 2: The architecture of AutoQ. The input verification problem is $\{\mathcal{P}, \varphi\}\ C\ \{\mathcal{Q}\}$.

Preprocessor reads the input files (`.aut`, `.smt` , `.qasm`, and `.hsl` files), translates specifications in the `.hsl` files into tree automata, and stores them using AutoQ's internal data structures. *Circuit Executor* then reads the circuit C and the TA $\mathcal{P}$ and generates another TA $\mathcal{R}$ obtained as the result after executing C from states in $\mathcal{P}$, using the approach of [14] with the symbolic extension discussed in Section 2. AutoQ can also output the TA $\mathcal{R}$ for further analysis. Finally, *Entailment Checker* checks whether $\mathcal{R} \models_\varphi \mathcal{Q}$ and reports "verified" when the entailment holds and "bug found" otherwise.

5 Use Cases

In this section, we describe several use cases of quantum algorithms and their important properties that we were able to verify using AutoQ fully automatically. We focus on the use of symbolic TA in this set of experiments and refer the readers to [14] for other experimental results. A selection of the obtained results is given in Table 1. An artifact that allows reproduction of the results is available as [13].

5.1 Hadamard Square is Identity

Our first use case shows that the single qubit gate C that runs two consecutive H gates has the same effect as an identity matrix. We use the specification $\{\mathcal{P}, \varphi\}\ C\ \{\mathcal{Q}\}$ with

$$\mathcal{P}\colon (|0\rangle\colon (v_a, v_b, v_c, v_d), |1\rangle\colon (v_a', v_b', v_c', v_d')), \qquad \varphi\colon \mathit{true},$$
$$\mathcal{Q}\colon (|0\rangle\colon (\Box_a, \Box_b, \Box_c, \Box_d) = (v_a, v_b, v_c, v_d), |1\rangle\colon (\Box_a, \Box_b, \Box_c, \Box_d) = (v_a', v_b', v_c', v_d')).$$

In this simple example, the precondition $\mathcal{P}$ encodes an infinite number of quantum states, which is not expressible using the technique in [14]. We also included a buggy version by altering one of the H gates, and AutoQ managed to detect the injected bug. The results can be found in rows H^2 in Table 1.

5.2 Zero Imaginary Part of Amplitudes

One property, which is shared by multiple algorithms, e.g., Bernstein-Vazirani's [8] and Grover's algorithm [25], is that the imaginary part of all amplitudes of the result is zero.

Let us focus on Bernstein-Vazirani's algorithm [8], which finds a secret bit-string s from an oracle using a single query. The algorithm begins with the quantum state $|0^n\rangle$, where n is the length of s, and ends with the quantum state $|s\rangle$. The amplitudes of all basis states are either zero or one, the imaginary part of the amplitudes is, therefore, always zero. For a three-qubit circuit C implementing the algorithm, we can therefore use the specification: $\{\mathcal{P}, \varphi\}\ C\ \{\mathcal{Q}\}$ with

$$\mathcal{P}\colon (|000\rangle\colon (1,0,0,0), |*\rangle\colon (0,0,0,0)), \qquad \varphi\colon \mathit{true}, \qquad \mathcal{Q}\colon (|*\rangle\colon \psi_{\mathrm{Im}}),$$

where $\psi_{\mathrm{Im}} \equiv (\Box_b = -\Box_d \wedge \Box_c = 0)$ (it will also be used later). In the definition of $\mathcal{P}$, recall that we use the integer-quadruple representation of complex numbers (cf. Eq. (5)). In the postcondition $\mathcal{Q}$, the free variables $\Box_a, \Box_b, \Box_c, \Box_d$ are to be substituted by the corresponding terms in the obtained integer term quadruple (a, b, c, d) in the entailment check. Note that (a, b, c, d) represents the complex number $(a + b\frac{\sqrt{2}}{2} - d\frac{\sqrt{2}}{2}) + i(b\frac{\sqrt{2}}{2} - c + d\frac{\sqrt{2}}{2})$ (obtained from Eq. (5)). Because a, b, c, d are all integers, for the imaginary part to be zero, it must hold that $c = 0$ and $b = -d$.

When we run C from $\mathcal{P}$, we obtain a TA $\mathcal{R}$ encoding $(|010\rangle\colon (1,0,0,0), |*\rangle\colon (0,0,0,0))$ and the entailment $\mathcal{R} \models_{\varphi} \mathcal{Q}$ holds. See the rows BV(n) in Table 1 for the results of verifying the algorithm for circuits with secrets of size n. As in the previous example, we also included a buggy version to demonstrate AutoQ's bug-finding capability. We can see that AutoQ could verify the algorithm for secrets of a quite large size.

5.3 Probability of Measuring the Correct Answer

Grover's algorithm [25] assumes a Boolean function f over n bits with only one satisfying assignment s and an oracle that evaluates f for a given input. The algorithm finds s with a high probability, say > 0.9, using only $\mathcal{O}(\sqrt{2^n})$ oracle queries. The algorithm works iteratively, where each *Grover iteration* queries the oracle once and amplifies the amplitude of $|s\rangle$. First, let C be a 6-qubit circuit implementing Grover's search with the satisfying assignment $s = 010$, where the first three qubits of C are the work tape, and the following three are the ancillae. We use the following specification:

$$\mathcal{P}\colon (|000000\rangle\colon \vec{1}, |*\rangle\colon \vec{0}) \text{ where } \vec{1} = (1,0,0,0) \text{ and } \vec{0} = (0,0,0,0), \qquad \varphi\colon \mathit{true},$$
$$\mathcal{Q}\colon (|010\rangle\colon |\Box_a|^2 > 0.9 \wedge \psi_{\mathrm{Im}}, |*\rangle\colon |\Box_a|^2 < 0.1 \wedge \psi_{\mathrm{Im}}) \otimes (|000\rangle\colon \vec{1}, |*\rangle\colon \vec{0}).$$

Table 1: Results of verifying our use cases with AutoQ. The maximum peak memory consumption was 52 MiB for $\text{Grover}_{\text{All}}(9)$. In most cases, the time of entailment was negligible, with the exception of $\text{Grover}_{\text{All}}$ circuits. For instance, $\text{Grover}_{\text{All}}(8)$ takes 2m18s for entailment checking (70% of the total time) and $\text{Grover}_{\text{All}}(9)$ takes 21m36s for entailment checking (85% of the total time).

circuit	qubits	gates	property	result	time	circuit	qubits	gates	property	result	time
H^2	1	2	$\text{H}^2 = \text{I}$	OK	0.22s	$\text{Grover}_{\text{Single}}(3)$	6	54	P(Correct) > 0.9	OK	0.34s
H^2 (bug)	1	2	$\text{H}^2 = \text{I}$	Bug	0.17s	$\text{Grover}_{\text{Single}}(16)$	32	28,159	P(Correct) > 0.9	OK	2m21s
BV(2)	2	6	ψ_{Im}	OK	0.11s	$\text{Grover}_{\text{Single}}(18)$	36	63,537	P(Correct) > 0.9	OK	6m37s
BV(2) (bug)	2	6	ψ_{Im}	Bug	0.15s	$\text{Grover}_{\text{Single}}(20)$	40	141,527	P(Correct) > 0.9	OK	19m57s
BV(100)	100	251	ψ_{Im}	OK	10.90s	$\text{Grover}_{\text{Iter}}(2)$	3	13	P(Correct) Increased	OK	0.40s
BV(1,000)	1,000	2,500	ψ_{Im}	OK	198m28s	$\text{Grover}_{\text{Iter}}(18)$	36	157	P(Correct) Increased	OK	1.95s
$\text{Grover}_{\text{All}}(3)$	9	64	P(Correct) > 0.9	OK	0.40s	$\text{Grover}_{\text{Iter}}(50)$	100	445	P(Correct) Increased	OK	47.76s
$\text{Grover}_{\text{All}}(8)$	24	939	P(Correct) > 0.9	OK	3m18s	$\text{Grover}_{\text{Iter}}(75)$	150	671	P(Correct) Increased	OK	3m29s
$\text{Grover}_{\text{All}}(9)$	27	1,492	P(Correct) > 0.9	OK	25m16s	$\text{Grover}_{\text{Iter}}(100)$	200	895	P(Correct) Increased	OK	10m53s

Note that the postcondition $\mathcal{Q}$ also checks that all amplitudes in the result of the algorithm have a zero imaginary part (using ψ_{Im}). See rows $\text{Grover}_{\text{Single}}(n)$ in Table 1 for the results on circuits for n-bit functions f and a single oracle.

Next, we also show the correctness of Grover's algorithm w.r.t. all possible 3-qubit oracles. Let C' be a 9-qubit circuit implementing the algorithm, where the first three qubits are used for oracle generation, and the following six are the work tape and ancillae, similarly to $\text{Grover}_{\text{Single}}$. Our specification is now

$$\begin{aligned}
&\mathcal{P}\colon \exists i \in \{0,1\}^3\colon (|i000000\rangle\colon \vec{1}, |*\rangle\colon \vec{0}), && \varphi\colon \mathit{true}, \\
&\mathcal{Q}\colon \exists i \in \{0,1\}^3\colon (|i\rangle\colon \vec{1}, |*\rangle\colon 0) \otimes (|i\rangle\colon |\Box_a|^2 > 0.9 \wedge \psi_{\text{Im}}, |*\rangle\colon |\Box_a|^2 < 0.1 \wedge \psi_{\text{Im}}) \\
&\qquad\qquad\qquad\qquad \otimes (|000\rangle\colon \vec{1}, |*\rangle\colon \vec{0}).
\end{aligned}$$

Note that in the postcondition, we use i to relate the oracle value and the value on the work tape. The results are in rows $\text{Grover}_{\text{All}}(n)$ in Table 1.

5.4 Increasing Amplitude of the Correct Answer

Above, we show that we are able to automatically verify moderate-sized circuits for Grover's algorithm for the values of n up to 9 (for $\text{Grover}_{\text{All}}$) and 20 (for $\text{Grover}_{\text{Single}}$), which is quite large, but have difficulties going beyond that. The size of the circuit is $\mathcal{O}(\sqrt{2^n})$, which is quite large. Therefore, we also verify the algorithm w.r.t. a weaker property, which is, that in one iteration, the amplitude of the correct answer will increase.

Consider a function f over 2 bits with 01 being the only satisfying assignment and let C be a 4-qubit circuit encoding one Grover iteration, with two qubits as the work tape and two ancilla qubits. From Grover's correctness proof [25], we can derive that when $v_\ell > 0 \wedge v_h > 0 \wedge (2^n - 1)v_\ell > v_h$, a correct implementation will increase the probability of $|01\rangle$ and reduce others. We specify the verification problem as follows:

$$\begin{aligned}
&\mathcal{P}\colon (|01\rangle\colon (v_h, 0, 0, 0), |*\rangle\colon (v_\ell, 0, 0, 0)) \otimes (|00\rangle\colon \vec{1}, |*\rangle\colon \vec{0}), \\
&\varphi\colon v_\ell > 0 \wedge v_h > 0 \wedge (2^2 - 1)v_\ell > v_h, \\
&\mathcal{Q}\colon (|01\rangle\colon |\Box_a| > |v_h| \wedge \psi_{\text{Im}}, |*\rangle\colon |\Box_a| < |v_\ell| \wedge \psi_{\text{Im}}) \otimes (|00\rangle\colon \vec{1}, |*\rangle\colon \vec{0}).
\end{aligned}$$

The results can be found in rows $\text{Grover}_{\text{Iter}}(n)$ in Table 1. We can see that verification of one Grover iteration w.r.t. the weaker (but still quite useful) property scales much better than verification of full Grover's circuits, scaling to sizes of $n \geq 100$.

6 Conclusion

We presented a specification language for specifying useful properties of quantum circuits and a tool AutoQ that can establish the correctness of the specification using an approach combining the technique from [14] with symbolic execution. Using the tool, we were able to fully automatically verify several important properties of a selection of quantum circuits. To the best of our knowledge, for some of the properties, we are the first ones that could verify them fully automatically.

Acknowledgements. We thank the reviewers for their useful remarks that helped us improve the quality of the paper. This work was supported by the Czech Ministry of Education, Youth and Sports project LL1908 of the ERC.CZ programme, the Czech Science Foundation project GA23-07565S, the FIT BUT internal project FIT-S-23-8151, and the NSTC QC project under Grant no. NSTC 111-2119-M-001-004- and 112-2119-M-001-006-.

References

1. Aws braket, https://aws.amazon.com/braket/
2. IBM quantum, https://quantum-computing.ibm.com
3. The Qcec repository: Issue #200 (ZX-Checker produces invalid result) (2022), https://github.com/cda-tum/qcec/issues/200
4. Abdulla, P.A., Chen, Y., Holík, L., Mayr, R., Vojnar, T.: When simulation meets antichains. In: Esparza, J., Majumdar, R. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6015, pp. 158–174. Springer (2010). https://doi.org/10.1007/978-3-642-12002-2_14, https://doi.org/10.1007/978-3-642-12002-2_14
5. Amy, M.: Towards large-scale functional verification of universal quantum circuits. In: Quantum Physics and Logic (2018)
6. Anticoli, L., Piazza, C., Taglialegne, L., Zuliani, P.: Towards quantum programs verification: From Quipper circuits to QPMC. In: Devitt, S.J., Lanese, I. (eds.) Reversible Computation - 8th International Conference, RC 2016, Bologna, Italy, July 7-8, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9720, pp. 213–219. Springer (2016). https://doi.org/10.1007/978-3-319-40578-0_16, https://doi.org/10.1007/978-3-319-40578-0_16
7. Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J.C., Barends, R., Biswas, R., Boixo, S., Brandao, F.G.S.L., Buell, D.A., Burkett, B., Chen, Y., Chen, Z., Chiaro, B., Collins, R., Courtney, W., Dunsworth, A., Farhi, E., Foxen, B., Fowler, A., Gidney, C., Giustina, M., Graff, R., Guerin, K., Habegger, S., Harrigan, M.P., Hartmann, M.J., Ho, A., Hoffmann, M., Huang, T., Humble, T.S., Isakov, S.V., Jeffrey, E., Jiang, Z., Kafri, D., Kechedzhi, K., Kelly, J., Klimov, P.V., Knysh, S., Korotkov, A., Kostritsa, F., Landhuis, D., Lindmark,

M., Lucero, E., Lyakh, D., Mandrà, S., McClean, J.R., McEwen, M., Megrant, A., Mi, X., Michielsen, K., Mohseni, M., Mutus, J., Naaman, O., Neeley, M., Neill, C., Niu, M.Y., Ostby, E., Petukhov, A., Platt, J.C., Quintana, C., Rieffel, E.G., Roushan, P., Rubin, N.C., Sank, D., Satzinger, K.J., Smelyanskiy, V., Sung, K.J., Trevithick, M.D., Vainsencher, A., Villalonga, B., White, T., Yao, Z.J., Yeh, P., Zalcman, A., Neven, H., Martinis, J.M.: Quantum supremacy using a programmable superconducting processor. Nature **574**(7779), 505–510 (Oct 2019). https://doi.org/10.1038/s41586-019-1666-5, https://www.nature.com/articles/s41586-019-1666-5, number: 7779 Publisher: Nature Publishing Group

8. Bernstein, E., Vazirani, U.V.: Quantum complexity theory. In: Kosaraju, S.R., Johnson, D.S., Aggarwal, A. (eds.) Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA. pp. 11–20. ACM (1993). https://doi.org/10.1145/167088.167097, https://doi.org/10.1145/167088.167097
9. Bertot, Y., Castéran, P.: Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions. Springer Science & Business Media (2013)
10. Bouajjani, A., Habermehl, P., Holík, L., Touili, T., Vojnar, T.: Antichain-based universality and inclusion testing over nondeterministic finite tree automata. In: Ibarra, O.H., Ravikumar, B. (eds.) Implementation and Applications of Automata, 13th International Conference, CIAA 2008, San Francisco, California, USA, July 21-24, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5148, pp. 57–67. Springer (2008). https://doi.org/10.1007/978-3-540-70844-5_7, https://doi.org/10.1007/978-3-540-70844-5_7
11. Burgholzer, L., Wille, R.: Advanced equivalence checking for quantum circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **40**(9), 1810–1824 (2020)
12. 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. Lecture Notes in Computer Science, vol. 12648, pp. 148–177. Springer International Publishing, Cham (March 2021)
13. Chen, Y., Chung, K., Lengál, O., Lin, J., Tsai, W.: AutoQ: An automata-based quantum circuit verifier (May 2023). https://doi.org/10.5281/zenodo.7966542, https://doi.org/10.5281/zenodo.7966542
14. Chen, Y., Chung, K., Lengál, O., Lin, J., Tsai, W., Yen, D.: An automata-based framework for verification and bug hunting in quantum circuits. In: 44th ACM SIGPLAN Conference on Programming Language Design and Implementation—PLDI'23. ACM (2023)
15. Coecke, B., Duncan, R.: Interacting quantum observables: categorical algebra and diagrammatics. New Journal of Physics **13**(4), 043016 (apr 2011). https://doi.org/10.1088/1367-2630/13/4/043016, https://doi.org/10.1088%2F1367-2630%2F13%2F4%2F043016
16. Coppersmith, D.: An approximate Fourier transform useful in quantum factoring (2002). https://doi.org/10.48550/arxiv.quant-ph/0201067, https://arxiv.org/abs/quant-ph/0201067
17. Cross, A.W., Bishop, L.S., Smolin, J.A., Gambetta, J.M.: Open quantum assembly language. arXiv preprint arXiv:1707.03429 (2017)
18. Dawson, C.M., Nielsen, M.A.: The Solovay-Kitaev algorithm. arXiv preprint quant-ph/0505030 (2005)
19. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings 14. pp. 337–340. Springer (2008)
20. D'Hondt, E., Panangaden, P.: Quantum weakest preconditions. Mathematical Structures in Computer Science **16**(3), 429–451 (2006)

21. Feng, Y., Hahn, E.M., Turrini, A., Zhang, L.: QPMC: A model checker for quantum programs and protocols. In: Bjørner, N., de Boer, F. (eds.) International Symposium on Formal Methods. pp. 265–272. Springer International Publishing (2015)
22. Feng, Y., Ying, M.: Quantum Hoare logic with classical variables. ACM Transactions on Quantum Computing **2**(4), 1–43 (2021)
23. Feng, Y., Yu, N., Ying, M.: Model checking quantum Markov chains. J. Comput. Syst. Sci. **79**(7), 1181–1198 (2013). https://doi.org/10.1016/j.jcss.2013.04.002, https://doi.org/10.1016/j.jcss.2013.04.002
24. Filliâtre, J.C., Paskevich, A.: Why3—where programs meet provers. In: Programming Languages and Systems: 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings 22. pp. 125–128. Springer (2013)
25. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Miller, G.L. (ed.) Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996. pp. 212–219. ACM (1996). https://doi.org/10.1145/237814.237866, https://doi.org/10.1145/237814.237866
26. Hietala, K., Rand, R., Hung, S.H., Wu, X., Hicks, M.: Verified optimization in a quantum intermediate representation. arXiv preprint arXiv:1904.06319 (2019)
27. King, J.C.: Symbolic execution and program testing. Commun. ACM **19**(7), 385–394 (1976). https://doi.org/10.1145/360248.360252, https://doi.org/10.1145/360248.360252
28. Lengál, O., Šimáček, J., Vojnar, T.: VATA: A library for efficient manipulation of non-deterministic tree automata. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 79–94. Springer (2012)
29. Liu, J., Zhan, B., Wang, S., Ying, S., Liu, T., Li, Y., Ying, M., Zhan, N.: Formal verification of quantum algorithms using quantum Hoare logic. In: International conference on computer aided verification. pp. 187–207. Springer (2019)
30. Mateus, P., Ramos, J., Sernadas, A., Sernadas, C.: Temporal Logics for Reasoning about Quantum Systems, p. 389–413. Cambridge University Press (2009). https://doi.org/10.1017/CBO9781139193313.011
31. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press, USA, 10th edn. (2011)
32. Perdrix, S.: Quantum entanglement analysis based on abstract interpretation. In: International Static Analysis Symposium. pp. 270–282. Springer (2008)
33. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994. pp. 124–134. IEEE Computer Society (1994). https://doi.org/10.1109/SFCS.1994.365700, https://doi.org/10.1109/SFCS.1994.365700
34. Tsai, Y., Jiang, J.R., Jhang, C.: Bit-slicing the Hilbert space: Scaling up accurate quantum circuit simulation. In: 58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021. pp. 439–444. IEEE (2021). https://doi.org/10.1109/DAC18074.2021.9586191, https://doi.org/10.1109/DAC18074.2021.9586191
35. Unruh, D.: Quantum Hoare logic with ghost variables. In: 2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). pp. 1–13. IEEE (2019)
36. Veanes, M., Bjørner, N.S.: Symbolic tree automata. Inf. Process. Lett. **115**(3), 418–424 (2015). https://doi.org/10.1016/j.ipl.2014.11.005, https://doi.org/10.1016/j.ipl.2014.11.005
37. Wenzel, M., Paulson, L.C., Nipkow, T.: The Isabelle framework. In: Theorem Proving in Higher Order Logics: 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings 21. pp. 33–38. Springer (2008)

38. Xu, M., Fu, J., Mei, J., Deng, Y.: Model checking QCTL plus on quantum Markov chains. Theor. Comput. Sci. **913**, 43–72 (2022). https://doi.org/10.1016/j.tcs.2022.01.044, https://doi.org/10.1016/j.tcs.2022.01.044
39. Xu, M., Li, Z., Padon, O., Lin, S., Pointing, J., Hirth, A., Ma, H., Palsberg, J., Aiken, A., Acar, U.A., et al.: Quartz: superoptimization of quantum circuits. In: Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation. pp. 625–640 (2022)
40. Ying, M.: Floyd-Hoare logic for quantum programs. ACM Transactions on Programming Languages and Systems (TOPLAS) **33**(6), 1–49 (2012)
41. Ying, M.: Model checking for verification of quantum circuits. In: International Symposium on Formal Methods. pp. 23–39. Springer (2021)
42. Ying, M., Feng, Y.: Model Checking Quantum Systems: Principles and Algorithms. Cambridge University Press (2021)
43. Yu, N., Palsberg, J.: Quantum abstract interpretation. In: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. pp. 542–558 (2021)
44. Zhou, L., Yu, N., Ying, M.: An applied quantum Hoare logic. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 1149–1162 (2019)
45. Zulehner, A., Hillmich, S., Wille, R.: How to efficiently handle complex values? implementing decision diagrams for quantum computing. In: Pan, D.Z. (ed.) Proceedings of the International Conference on Computer-Aided Design, ICCAD 2019, Westminster, CO, USA, November 4-7, 2019. pp. 1–7. ACM (2019). https://doi.org/10.1109/ICCAD45719.2019.8942057, https://doi.org/10.1109/ICCAD45719.2019.8942057
46. Zulehner, A., Wille, R.: Advanced simulation of quantum computations. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **38**(5), 848–859 (2019). https://doi.org/10.1109/TCAD.2018.2834427, https://doi.org/10.1109/TCAD.2018.2834427