A Practical Specification Language for Automatic Quantum Program Verification

Wei-Lun Tsai[1,2] ✉, Yu-Fang Chen[1], and Ondřej Lengál[3]

[1] Institute of Information Science, Academia Sinica, Taiwan
[2] Graduate Institute of Electronics Engineering, National Taiwan University, Taiwan
[3] Faculty of Information Technology, Brno University of Technology, Czech Republic
alan23273850@gmail.com, gulu0724@gmail.com, lengal@fit.vutbr.cz

Abstract. Hoare-style verification provides a principled foundation for reasoning about the correctness of quantum programs, but existing approaches do not allow fully automatic verification. While automata-based verification scales well when specifications are given directly as automata, prior frameworks incur exponential blow-up when translating high-level set-based assertions into automata, which severely limits practicality. We introduce an extended set-based specification language and a specification-to-automata translation algorithm whose complexity is linear in the number of qubits, enabled by controlled automaton construction and qubit reordering. The resulting compact automata enable fully automatic Hoare-style verification of fixed-qubit quantum programs at previously infeasible scales, while substantially improving expressiveness without compromising efficiency.

1 Introduction

Recent advances in quantum hardware, exemplified by early demonstrations targeting quantum supremacy [4] and evidence of quantum utility on processors exceeding 100 qubits [24], have accelerated the development of quantum programming. This progress is shifting the field from predominantly theoretical studies toward practical applications in cryptography [29], finance [31], and optimization [23]. As quantum programs grow in size and complexity, ensuring their correctness becomes increasingly critical.

Despite significant progress in quantum program logics and verification frameworks in the last decade [1, 3, 10–17, 39, 43], a gap remains between theory and practice. In particular, most existing approaches do not provide a practical specification language that allows users to state nontrivial correctness properties and have them verified fully automatically, without interactive proof construction. Bridging this gap is essential for making formal verification a usable component of the quantum software development workflow. *Quantum Hoare logic* (QHL) is a widely used framework for reasoning about the correctness of quantum programs. In QHL, program behavior is specified using triples $\{P\}\, C\, \{Q\}$, where the precondition P and postcondition Q are represented as *Hermitian operators*

over the underlying Hilbert space [39]. In the past, a variety of quantum Hoare-style systems have been developed [19, 20, 25, 27, 30, 32, 36–39, 41, 43], providing sound semantic foundations for formal reasoning about program correctness.

However, existing quantum Hoare-style approaches do not yet support practical push-button verification. Writing assertions that capture nontrivial correctness properties typically leads to complex proof obligations that require substantial manual effort in interactive theorem provers. For example, even for Grover's search algorithm [22], formal verification for an arbitrary number of qubits has only been achieved through large, hand-crafted proof developments in systems such as Isabelle/HOL and Coq, comprising hundreds to thousands of lines of proof scripts [28, 42]. The limitations of existing quantum Hoare-style verification frameworks suggest that the choice of assertion representation plays a decisive role in enabling automation. In this work, we advocate assertions based on *sets of quantum states* as a powerful and natural alternative.

Set-based assertions provide a natural and expressive way to specify correctness properties of quantum programs. Unlike traditional assertion representations, they can directly describe families of quantum states that satisfy explicit quantitative constraints. For example, the postcondition of Grover's algorithm can be specified as the set $\bigcup_{|\alpha|^2 > 0.8} \left\{ \alpha \left|w\right\rangle + \beta \sum_{i \neq w} \left|i\right\rangle \right\}$ for some $n \geq 2$ [40] and marked item $w \in \{0,1\}^n$. This specification directly captures the intended probabilistic guarantee without resorting to indirect encodings or symbolic reasoning about amplitudes. Such specifications are particularly well-suited to automated verification: correctness properties are stated as concrete constraints over sets of states, which in principle can be manipulated algorithmically. Building on this idea, AutoQ [13] represents set-based assertions using finite automata and SMT constraints, and performs Hoare-style verification through algorithmic transformations over these representations. Once the specification is given, the verification process proceeds without interactive theorem proving.

However, the practical applicability of this approach is limited by the cost of translating specifications into automata. In prior work, the size of the generated automata can grow exponentially with respect to the number of qubits. This exponential blow-up severely restricts verification to very small instances, despite the conceptual suitability of set-based assertions for automation. In this work, we address this scalability bottleneck by introducing a new specification-to-automata translation algorithm whose complexity is linear in the qubit number. Before building automata, it reorganizes the specification into a tensor product of smaller, mostly independent components (first at the variable level, then at the qubit level), and only then constructs and composes the corresponding automata. As a result, extended set-based specifications can be compiled into compact automata even for nontrivial cases, enabling fully automatic verification at a scale that was previously infeasible.

We implemented the new translation algorithm and compared it against the method in [13]. The results show that our algorithm substantially outperforms the prior approach. For instance, we translate the functional-correctness spec-

ification of a 32-qubit Grover circuit into an automaton in under one second, whereas [13] does not complete the translation within five minutes.

Related Work. Within quantum Hoare logic, two principal representations of predicates have emerged. The first represents assertions as Hermitian operators, following the work of D'Hondt and Panangaden [18] and further developed by Ying [39]. This formulation enables quantitative reasoning over mixed states, supporting properties such as success probabilities and expected values. The second represents assertions as projections, or closed subspaces of Hilbert spaces, rooted in the quantum logic of Birkhoff and von Neumann [7] and later applied to quantum Hoare logic for qualitative reasoning [43]. While both representations are mathematically well-founded, they pose challenges for automation. The implementation is often via interactive theorem provers, and the proof requires significant manual efforts [28, 42].

Set-based assertions have recently been explored as an alternative specification mechanism, aiming to better support automation by treating correctness properties as explicit sets of quantum states. Prior work has shown that such assertions can be verified algorithmically when given directly as automata representations [1,2,13–15]. However, existing approaches suffer from severe scalability issues when translating high-level specifications into automata, often incurring exponential blow-up.

2 Background

2.1 Quantum Computing

In quantum computing, an n-qubit *quantum state* is a superposition of all 2^n computational basis states: $|\psi\rangle = \sum_{i\in\{0,1\}^n} c_i\,|i\rangle$, where each complex *amplitude* c_i is associated with the basis state $|i\rangle$ and satisfies the normalization condition $\sum_i |c_i|^2 = 1$. For example, $\frac{1}{2}\,|00\rangle - \frac{1}{2}\,|01\rangle + \frac{i}{\sqrt{2}}\,|10\rangle$ is a valid two-qubit state.

A quantum state can be viewed structurally as a *perfect binary tree* of height n. The k-th level corresponds to the k-th qubit, and each basis string in $\{0,1\}^n$ determines a unique root-to-leaf path, taking the left branch for 0 and the right branch for 1. The leaf reached by this path stores the amplitude of the corresponding basis state. See Figures 1(a) and 1(b) for illustrations.

To compose quantum systems, let $|\psi\rangle = \sum_{s\in\{0,1\}^n} a_s\,|s\rangle$ be an n-qubit state and $|\phi\rangle = \sum_{t\in\{0,1\}^m} b_t\,|t\rangle$ an m-qubit state. Their tensor product is the $(n+m)$-qubit state $|\psi\rangle \otimes |\phi\rangle = \sum_{s\in\{0,1\}^n, t\in\{0,1\}^m} a_s b_t\,|st\rangle$, where $|st\rangle$ denotes concatenation of basis strings. In the tree representation, this corresponds to replacing each leaf labeled a_s in the tree of $|\psi\rangle$ by a copy of the tree of $|\phi\rangle$, scaled by a_s. We extend this operation elementwise to sets of quantum states. For sets S_1 and S_2, define $S_1 \otimes S_2 = \{|\psi\rangle \otimes |\phi\rangle \mid |\psi\rangle \in S_1,\ |\phi\rangle \in S_2\}$.

Finally, quantum computation proceeds by applying *quantum gates*, which are unitary operators mapping quantum states to quantum states while preserving normalization. Quantum gates are the fundamental building blocks of *quantum circuits*.

2.2 Level-Synchronized Tree Automata

We must choose an automata model as the target formalism for specification translation. We adopt *Level-Synchronized Tree Automata* (LSTA) [1]. Compared with standard tree automata [14,15], LSTAs provide a more compact encoding of quantum states and are directly supported by the verification tool AutoQ [12,13]. By translating specifications into LSTAs, we can directly reuse AutoQ's decision procedures and obtain a fully automatic, end-to-end verification workflow. Thus, LSTAs are not only expressive and succinct, but also practically well suited for tool-supported verification. As discussed in Section 2.1, a quantum state can be viewed structurally as a perfect binary tree. LSTAs compactly represent *sets* of such trees by sharing common substructures. This section introduces the formal definition of LSTAs, their semantics, and supported operations.

Definition. Let $\mathbb{K}$ be a commutative nonunital semiring (i.e., a structure closed under addition and multiplication, equipped with an additive identity $0_{\mathbb{K}}$ that is absorbing for multiplication). A *level-synchronized tree automaton (LSTA)* [1] over $\mathbb{K}$ is a tuple $\mathcal{A} = \langle Q, V, \Delta, r \rangle_{\mathbb{K}}$ where Q is a set of *states*, V is a set of *variables*, Δ is a set of *transitions*, and $r \in Q$ is the *root state* (or *starting state*). We often omit the subscript $\mathbb{K}$ when the semiring is clear from the context. The transition set Δ is divided into two disjoint nonempty subsets: Δ_{in} (*internal transitions*) and Δ_{ex} (*external or leaf transitions*).

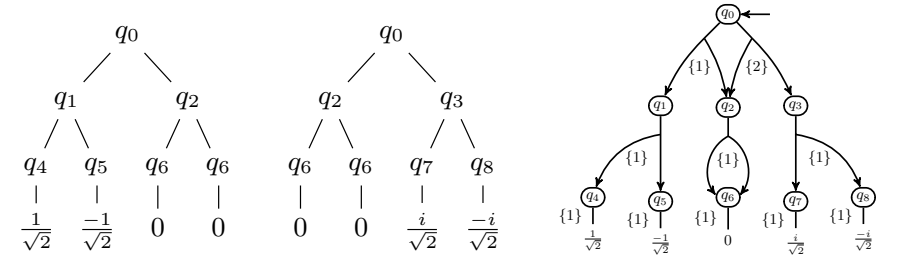
An internal transition $\delta \in \Delta_{in}$ has the form $q \xrightarrow{C} (q_1, q_2)$, and a leaf transition $\delta \in \Delta_{ex}$ has the form $q \xrightarrow{C} e$, where $q, q_1, q_2 \in Q$, $e \in \mathbb{K}[V]$ (i.e., a polynomial over V with coefficients in $\mathbb{K}$), and $C \subseteq \mathbb{N}$ is a nonempty finite set of *choices*. We refer to q, q_1, q_2, e, and C as the *top state*, *left child*, *right child*, *amplitude*, and *choices* of the transition δ, and denote them as $\texttt{top}(\delta)$, $\texttt{left}(\delta)$, $\texttt{right}(\delta)$, $\texttt{amp}(\delta)$, and $\texttt{ch}(\delta)$, respectively.

We define *root transitions* as $\Delta_r = \{\delta \in \Delta_{in} \mid \texttt{top}(\delta) = r\}$. To ensure deterministic resolution of transitions, LSTAs satisfy *choice disjointness*: for any two distinct transitions δ_1, δ_2 with the same top state, their choice sets are disjoint. The size of an LSTA $\mathcal{A}$, denoted $|\mathcal{A}|$, is defined as $|\Delta|$.

Sets of Quantum States. In this work, an LSTA is used as an internal representation of a set of quantum states. Intuitively, an LSTA $\mathcal{A}$ encodes a set $\mathcal{L}(\mathcal{A})$ of perfect binary trees, each corresponding to a quantum state as described in Section 2.1. A quantum state $|\psi\rangle = \sum_{s \in \{0,1\}^n} a_s |s\rangle$ belongs to $\mathcal{L}(\mathcal{A})$ if and only if there exists a sequence of choices $c_1, \ldots, c_n, c_0 \in \mathbb{N}$ that induces a perfect binary tree whose leaf amplitudes match the coefficients a_s. Such a tree, if it exists, is unique by choice disjointness.

Formally, the sequence of choices must satisfy the following conditions.

(1) The choices $c_1, \ldots, c_n, c_0$ induce a root-to-leaf path for every branch of the tree. For each basis string $s = b_1 \ldots b_n \in \{0,1\}^n$, there exists a unique sequence of states $q_0, q_1, \ldots, q_n$ and a value v_s such that $q_0 = r$ and: (a) for



(a) $\frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|01\rangle$ (b) $\frac{i}{\sqrt{2}}|10\rangle - \frac{i}{\sqrt{2}}|11\rangle$ (c) The LSTA representation

Fig. 1. Representing a set of quantum states with an LSTA

each internal level $1 \le i \le n$, there exists a unique internal transition $\delta \in \Delta_{in}$ with $\mathtt{top}(\delta) = q_{i-1} \ \wedge \ c_i \in \mathtt{ch}(\delta) \ \wedge \ ite(b_i = 0, \mathtt{left}(\delta) = q_i, \mathtt{right}(\delta) = q_i)$; (b) at the leaf level, there exists a unique external transition $\delta \in \Delta_{ex}$ with $\mathtt{top}(\delta) = q_n \ \wedge \ c_0 \in \mathtt{ch}(\delta) \ \wedge \ \mathtt{amp}(\delta) = v_s$.
A transition δ is said to be *enabled* by a choice c if $c \in \mathtt{ch}(\delta)$.

(2) For all $s \in \{0,1\}^n$, the induced value satisfies $v_s = a_s$.

Example 1. Consider a set of quantum states $\{\frac{1}{\sqrt{2}}(|00\rangle - |01\rangle), \frac{i}{\sqrt{2}}(|10\rangle - |11\rangle)\}$ and an LSTA $\langle Q, \emptyset, \Delta_{in} \cup \Delta_{ex}, r\rangle$, where $Q = \{q_i \mid 0 \le i \le 8\}$, $\Delta_{in} = \{q_0 \xrightarrow{\{1\}} (q_1, q_2),\ q_0 \xrightarrow{\{2\}} (q_2, q_3),\ q_1 \xrightarrow{\{1\}} (q_4, q_5),\ q_2 \xrightarrow{\{1\}} (q_6, q_6),\ q_3 \xrightarrow{\{1\}} (q_7, q_8)\}$, $\Delta_{ex} = \{\ q_4 \xrightarrow{\{1\}} \frac{1}{\sqrt{2}},\ q_5 \xrightarrow{\{1\}} \frac{-1}{\sqrt{2}},\ q_6 \xrightarrow{\{1\}} 0,\ q_7 \xrightarrow{\{1\}} \frac{i}{\sqrt{2}},\ q_8 \xrightarrow{\{1\}} \frac{-i}{\sqrt{2}}\}$, $r = q_0$. Figure 1 shows two quantum states as binary trees and their vertical LSTA representations to better illustrate the induction process. All transitions whose choices include 1 collectively form the tree in Figure 1(a), whereas replacing the root transition with the alternative one yields the tree in Figure 1(b). These are the only trees that can be induced by Figure 1(c).

Binary Operations. The *set union* and *tensor product* operations on LSTAs, corresponding to the semantics of language operations, serve as the fundamental building blocks for constructing the target LSTA. The upper bounds on the sizes of the constructed LSTAs, which are keys to the desired complexity, are summarized in the following theorem (see Appendix A.1 in [34] for the proof).

Theorem 2. *Given two LSTAs $\mathcal{A}$ and $\mathcal{B}$ over $\mathbb{K}$ where $\mathcal{L}(\mathcal{A})$ and $\mathcal{L}(\mathcal{B})$ contain n-qubit and m-qubit states, respectively, there exists a set union operation (denoted by $\mathcal{A} \sqcup \mathcal{B}$) and a tensor product operation (denoted by $\mathcal{A} \otimes \mathcal{B}$), both yielding valid LSTAs over $\mathbb{K}$. These operations satisfy the semantic properties $\mathcal{L}(\mathcal{A} \sqcup \mathcal{B}) = \mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{B})$ and $\mathcal{L}(\mathcal{A} \otimes \mathcal{B}) = \mathcal{L}(\mathcal{A}) \otimes \mathcal{L}(\mathcal{B})$. Regarding the automaton size, the set union is bounded by $|\mathcal{A} \sqcup \mathcal{B}| \le |\mathcal{A}| + |\mathcal{B}|$ and the tensor product is bounded by $|\mathcal{A} \otimes \mathcal{B}| \le |\mathcal{A}| + N_{leaves}(\mathcal{A}) \cdot |\mathcal{B}|$, where $N_{leaves}(\mathcal{A})$ denotes the number of distinct amplitude values at the leaves of $\mathcal{A}$.*

3 Specification Language

This section formally introduces the specification language used to describe *assertions* (i.e., sets of quantum states), which serve as preconditions and postconditions in our verification framework. Our design aligns with the Dirac notation and standard set representation, ensuring familiarity and expressiveness. The syntax is shown in Fig. 2, with *expr* as the start symbol. We define two disjoint sets of variable names for later use: V_s for binary string variables (used in basis states) and V_c for complex variables.

$$
\begin{aligned}
\mathit{expr} &::= \mathit{tset} \mid \bigcup_{\mathit{CCons}} \mathit{tset} \\
\mathit{tset} &::= \mathit{pset} \mid \mathit{tset} \otimes \mathit{pset} \\
\mathit{pset} &::= \mathit{uset} \mid \mathit{uset}^{N} \\
\mathit{uset} &::= \mathit{set} \mid \mathit{uset} \cup \mathit{set} \\
\mathit{set} &::= \{\mathit{diracs}\} \mid \{\mathit{diracs} : \mathit{varcons}\}
\end{aligned}
\qquad
\begin{aligned}
\mathit{diracs} &::= \mathit{dirac} \mid \mathit{dirac},\ \mathit{diracs} \\
\mathit{dirac} &::= \mathit{term} \mid \mathit{dirac} + \mathit{term} \\
\mathit{term} &::= \alpha\,|\mathit{VStr}\rangle \mid \alpha \sum_{\mathit{varcons}} |\mathit{VStr}\rangle \\
\mathit{varcons} &::= \mathit{varcon} \mid \mathit{varcon},\ \mathit{varcons} \\
\mathit{varcon} &::= |V| = N \mid V \neq V \mid V \neq \mathit{CStr} \mid V = \mathit{CStr}
\end{aligned}
$$

Fig. 2. Syntax of the specification language. The grammar comprises six terminal categories: natural number $N \in \mathbb{N}$; constant binary string $\mathit{CStr} \in \{0,1\}^{+}$; complex polynomial $\alpha \in \mathbb{C}[V_c]$; binary string variable $V \in V_s$; constraint CCons, defined as a quantifier-free first-order formula in nonlinear arithmetic over the real part and imaginary part of variables in V_c; and basis string pattern $\mathit{VStr} \in (\{0,1\} \cup V_s \cup \overline{V_s})^{+}$, where $\overline{V_s} = \{\overline{v} \mid v \in V_s\}$ collects bit-complemented variables.

The grammar is designed to possess three key features for quantum program verification:

- **Modular Construction:** Two fundamental operations allow for the construction of complex sets from simpler components: the standard set union ($\cup$) to aggregate states and the tensor product ($\otimes$) to compose quantum subsystems. Our syntax assigns $\otimes$ the lowest precedence among all other operators except $\bigcup_{\mathit{CCons}}$ to make $\otimes$ act as a natural structural delimiter that separates distinct subsystems in our algorithm. For instance, $A \cup B \otimes C$ is interpreted as $(A \cup B) \otimes C$.
- **Symbolic Representation:** Beyond concrete values, the grammar admits symbolic variables in both amplitudes (via V_c) and basis states (via V_s). The variables in V_c enable the specification of infinite sets of states. The variables in V_s can be used along with the summation ($\sum$) to describe superpositions compactly. This allows a single *term* to represent a linear combination of exponentially many basis states without explicit enumeration. For instance, $\left\{ \frac{1}{\sqrt{7}} \sum_{j \neq i} |j\rangle : |i| = 3 \right\}$ represents a set of normalized uniform superpositions where exactly one basis state $|i\rangle$ is excluded from the full basis.

– **Constraint-Based Specification:** The terminal *CCons* and nonterminal *varcons* enable precise control over the valid state space by applying constraints to variables. For instance, $\bigcup_{|\alpha|^2>0.8} \left\{ \alpha \left|w\right\rangle + \beta \sum_{i \neq w} \left|i\right\rangle \right\}$ for $n \geq 2$ and marked item $w \in \{0,1\}^n$ can be a postcondition of Grover's algorithm.

In addition, the grammar provides specific constructs to facilitate compact specifications. We denote the N-fold tensor product of a set S as S^N, inductively defined by $S^1 = S$ and $S^N = S^{N-1} \otimes S$. This is particularly useful for describing uniform registers, such as an N-qubit zero state $\{|0\rangle\}^N$. Furthermore, the constraint $|V| = N$ is used to explicitly define the domain of a variable V as $\{0,1\}^N$. Example usage is provided in Section 4.

Syntactic correctness does not guarantee semantic validity. To be considered *well-formed*, an *assertion* generated by the grammar must satisfy the following:

(1) **Unambiguous Variable Length:** For every binary string variable V in a *VStr*, its *length* N (i.e., the number of qubits) must be uniquely determined, either explicitly via the constraint $|V| = N$ or implicitly inferred from constraints such as $V \neq V'$, $V \neq CStr$, or $V = CStr$, provided that V' or *CStr* has a known length of N.
(2) **Length Consistency:** The number of qubits must be consistent across all quantum states within a single *uset*, as well as between inequality operands.
(3) **No Redundant Summation Variables:** Within each *term*, every *iterating* variable under the summation (if any) must be present in *VStr*. Otherwise, the summation results in unintended amplitude scaling, which is undesirable. For instance, $\left\{ \frac{1}{\sqrt{7}} \sum_{j \neq i} \left|j\right\rangle : |i| = 3 \right\}$ has no redundant summation variables because *VStr* contains the only iterating variable j, but $\left\{ \frac{1}{\sqrt{7}} \sum_{j \neq i,\ k \neq i,\ |\ell|=2} \left|j\right\rangle : |i| = 3 \right\}$ has two redundant variables k and ℓ, which results in an unintended scaling factor of $(2^3 - 1) \times 2^2 = 28$.

4 Use Cases

4.1 Oracle-Based Algorithms

An *oracle circuit* is a black-box circuit that encodes a function and enables quantum algorithms to query information in a single step. In Grover's search [22], the oracle implements $f(x) \colon \mathbb{B}^n \to \mathbb{B}$, returning 1 on the marked solution x and 0 otherwise; in Bernstein–Vazirani (BV) [6], it encodes a secret bit string. To verify such algorithms for *all* oracles, we use a *parameterized* oracle circuit whose behavior is determined by input qubits via controlled gates, and compose it with the circuit under verification.

For BV, the composed circuit has 7 qubits (Fig. 3): the highlighted block is the oracle, and the rest is the implementation. We treat the secret as part of the input: qubits s_1, s_2, s_3 parameterize the oracle, the remaining 0-qubits are workspace, and the last qubit is an ancilla. We prove correctness by showing $\{|s0001\rangle : |s| = 3\} \Rightarrow \{|ss1\rangle : |s| = 3\}$, i.e., for every secret string s, the circuit outputs s.

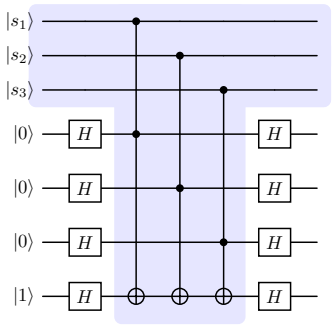


Fig. 3. 3-qubit BV circuit.

4.2 Amplitude Amplification

When verifying an amplitude amplification algorithm (e.g., Grover's search [22]), we can use variables v_h and v_ℓ as amplitudes and describe the relation between the variables before and after the circuit evolution using a global constraint. For a Grover iteration circuit where the marked state is $|111\rangle$ on the first three qubits and the remaining three qubits serve as ancilla, we verify its correctness using the following specification.

$$\bigcup_{\substack{\text{imag}(v_h)=0,\ \text{real}(v_h)>0,\\ \text{imag}(v_\ell)=0,\ \text{real}(v_\ell)>0,\\ 7v_\ell>v_h}} \{v_h\,|111001\rangle + v_\ell \sum_{i\neq 111} |i001\rangle\} \Rightarrow \bigcup_{\substack{\text{imag}(v'_h)=0,\\ \text{imag}(v'_\ell)=0,\\ |v'_h|>|v_h|}} \{v'_h\,|111001\rangle + v'_\ell \sum_{i\neq 111} |i001\rangle\},$$

where $P \Rightarrow Q$ means P is the *precondition* and Q is the *postcondition*. The language also allows us to specify the property that a complete Grover's circuit has $>80\,\%$ probability of finding the marked state as follows.

$$\{|000001\rangle\} \Rightarrow \bigcup_{|v_h|^2 > 0.8} \{v_h\,|111001\rangle + v_\ell \sum_{i\neq 111} |i001\rangle\}$$

4.3 Compound Multi-Control Quantum Gates

Quantum hardware typically supports only a limited gate set, so implementing an unsupported gate often requires decomposing it into a sequence of native gates. For example, an n-controlled Toffoli gate is usually realized using standard Toffoli gates. In Fig. 4, the qubits $c_1, \ldots, c_5$ are controls, the intermediate $|0\rangle$ registers are ancillas, and the final qubit $|t\rangle$ is the target. We verify correctness against the following specifications:

$\{|c00000\rangle : c = 11111\} \Rightarrow \{|c00001\rangle : c = 11111\}$
$\{|c00001\rangle : c = 11111\} \Rightarrow \{|c00000\rangle : c = 11111\}$
$\{|c00000\rangle : c \neq 11111\} \Rightarrow \{|c00000\rangle : c \neq 11111\}$
$\{|c00001\rangle : c \neq 11111\} \Rightarrow \{|c00001\rangle : c \neq 11111\}$

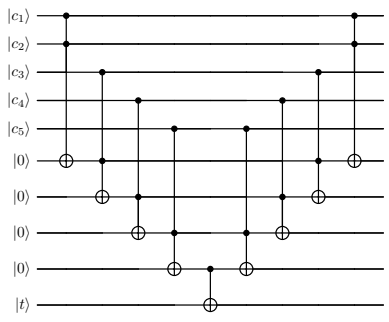


Fig. 4. Multi-controlled Toffoli with 5 control qubits. Standard Toffoli gates have two • as controls and one ⊕ as the target.

5 From Specification to Automata

5.1 Overview

This section details the algorithm that translates a list of input assertions into their corresponding automata, with qubits reordered strategically. The procedure employs a divide-and-conquer strategy to reduce construction complexity. First, we preprocess the input to **ensure the variable-boundary-aligned representation** (Section 5.2). Next, we perform a two-stage transformation: a **high-level variable reordering** (Section 5.3) followed by a **low-level qubit reordering** (Section 5.4). These steps aim to transform assertions into tensor products of smaller set components with qubits reordered, constituting a key contribution of this work. Finally, we **construct compact automata** for these components and recompose them using set union and tensor product operations to yield the final automaton (Section 5.5).

5.2 Ensuring Variable-Boundary-Aligned Representation

To facilitate the core translation algorithm, we first process the raw input assertions into variable-boundary-aligned representations through four steps.

1. **Canonicalization (Syntactic Sugar Elimination and Variable Renaming):** To streamline the core algorithm steps, we canonicalize the input representation in advance. We first eliminate two forms of syntactic sugar through rewriting. Specifically, a tensor power uset^N is expanded into the N-fold tensor product $\underbrace{\mathit{uset} \otimes \cdots \otimes \mathit{uset}}_{N}$, and a *set* of the form $\{\mathit{dirac}_1,\ \mathit{dirac}_2,\ \ldots,\ \mathit{dirac}_k : \mathit{varcons}\}$ is split into the k-fold union $\{\mathit{dirac}_1 : \mathit{varcons}\} \cup \{\mathit{dirac}_2 : \mathit{varcons}\} \cup \cdots \cup \{\mathit{dirac}_k : \mathit{varcons}\}$. This rule applies analogously to sets without *varcons*. After that, we perform alpha-renaming to ensure that distinct variables are assigned unique names, thereby streamlining the dependency analysis in Section 5.3. These transformations yield a form that is free of syntactic sugar and ensures variable uniqueness.
2. **Tensor Alignment Check:** We verify that all assertions after canonicalization contain the same number of tensor product operators ($\otimes$). If this check passes, we further decompose each assertion into a sequence of *uset* constructs delimited by these operators. By defining the collection of the i-th *uset* from all assertions as the i-th *tensor segment*, we then verify that within each segment, all quantum states possess the same qubit length. *Any mismatch in segment count or qubit length terminates the process.*
3. **Variable Alignment Check:** We verify that the boundaries of all variables are aligned. Operationally, this is implemented by checking that any two variables occupy either disjoint or identical qubit intervals. *Any violation aborts the process.*
4. **Constant Abstraction:** Upon passing the alignment checks, we compute the *global partition*, a set of disjoint qubit intervals that partitions the range

$[1,\ n+1)$ where n denotes the number of qubits in all assertions. The partition respects the boundaries of all variables, which means each interval $[a, b)$ occupied by a variable contributes exactly one element to the partition. To maintain structural consistency, any constant binary string spanning multiple intervals is sliced to match these boundaries. Each resulting slice is then abstracted into a fresh variable V bound to its corresponding constant value *CStr* via inserted equalities $V = CStr$ under the summation, ensuring that every rewritten *VStr* aligns with the global partition at the variable level.

Following this preprocessing phase, the original *set* constructs are transformed into restricted *setP* constructs, in which every *VStr* is free of constant binary digits.

Example 3 (Preprocessing Pipeline). Consider an input of two assertions.

$$\mathcal{E}_1 = \{\ |i\,0\,0\rangle\ :\ |i| = 2\ \} \otimes \{|0\rangle\} \cup \{|1\rangle\}^2$$
$$\mathcal{E}_2 = \{\ |00\,0\,i\rangle\,,\ |11\,1\,i\rangle\ :\ |i| = 1\ \} \otimes \{|0\rangle\} \otimes \{|0\rangle\}$$

Step 1 (Canonicalization): We rewrite $\mathcal{E}_1$ to eliminate the tensor power and rewrite $\mathcal{E}_2$ to eliminate the comma-separated list. Additionally, we resolve the naming conflict by alpha-renaming the variable i in $\mathcal{E}_2$ to j in the first term and k in the second term.

$$\mathcal{E}_1' = \underbrace{\{\ |i\,0\,0\rangle\ :\ |i| = 2\ \}}_{\text{Segment 1}} \otimes \underbrace{\{|0\rangle\} \cup \{|1\rangle\}}_{\text{Segment 2}} \otimes \underbrace{\{|0\rangle\} \cup \{|1\rangle\}}_{\text{Segment 3}}$$
$$\mathcal{E}_2' = \underbrace{\{\ |00\,0\,j\rangle\ :\ |j| = 1\ \} \cup \{\ |11\,1\,k\rangle\ :\ |k| = 1\ \}}_{\text{Segment 1}} \otimes \underbrace{\{|0\rangle\}}_{\text{Segment 2}} \otimes \underbrace{\{|0\rangle\}}_{\text{Segment 3}}$$

Step 2 (Tensor Alignment Check): We decompose both canonicalized assertions into segments delimited by tensor product operators. Since $\mathcal{E}_1'$ and $\mathcal{E}_2'$ both have three segments, the segment counts pass. We proceed to the qubit lengths. In Segment 1, $\mathcal{E}_1'$ terms have length $2+1+1 = 4$ and $\mathcal{E}_2'$ term has length $3 + 1 = 4$. In Segments 2 and 3, both $\mathcal{E}_1'$ terms and the $\mathcal{E}_2'$ term have length 1, so the qubit lengths also pass.

Step 3 (Variable Alignment Check): We check the variable boundaries for Segment 1. In $\mathcal{E}_1'$, variable i occupies $[1, 3)$. In $\mathcal{E}_2'$, the renamed variables j and k occupy $[4, 5)$. Since all intervals are either disjoint or identical, the variable boundaries are consistent across all segments and hence pass this check.

Step 4 (Constant Abstraction): The global partition $\mathcal{I}$ across all three segments is $\{\overbrace{\underbrace{[1,3)}_{\text{Slot 1}}}^{\text{Segment 1}}, \overbrace{\underbrace{[3,4)}_{\text{Slot 2}}}^{\text{Segment 1}}, \overbrace{\underbrace{[4,5)}_{\text{Slot 3}}}^{\text{Segment 1}}, \overbrace{\underbrace{[5,6)}_{\text{Slot 4}}}^{\text{Segment 2}}\ ,\ \overbrace{\underbrace{[6,7)}_{\text{Slot 5}}}^{\text{Segment 3}}\ \}$. We slice and abstract con-

stants to match $\mathcal{I}$.

$$\mathcal{E}_1'' = \underbrace{\{ \sum_{\substack{a=0,\\ b=0}} |iab\rangle : |i| = 2 \}}_{\text{Segment 1}} \otimes \underbrace{\{ \sum_{g=0} |g\rangle \}}_{\text{Segment 2}} \cup \{ \sum_{h=1} |h\rangle \} \otimes \underbrace{\{ \sum_{m=0} |m\rangle \} \cup \{ \sum_{n=1} |n\rangle \}}_{\text{Segment 3}}$$

$$\mathcal{E}_2'' = \underbrace{\{ \sum_{\substack{c=00,\\ d=0}} |cdj\rangle : |j| = 1 \} \cup \{ \sum_{\substack{e=11,\\ f=1}} |efk\rangle : |k| = 1 \}}_{\text{Segment 1}} \otimes \underbrace{\{ \sum_{\ell=0} |\ell\rangle \}}_{\text{Segment 2}} \otimes \underbrace{\{ \sum_{p=0} |p\rangle \}}_{\text{Segment 3}}$$

The final rewritten assertions $\mathcal{E}_1''$ and $\mathcal{E}_2''$ consist purely of variable-boundary-aligned terms, ready for transformation. In this aligned form, we refer to each interval in the global partition as a *slot* to highlight that each slot can be occupied by exactly one variable. In this case, there are five slots numbered from 1 to 5 in this specification, where Segment 1 occupies the first three slots, Segment 2 occupies the fourth slot, and Segment 3 occupies the fifth slot.

5.3 Variable-Level Reordering and Tensor Product Transformation

This section functions as a high-level complexity reduction strategy. It analyzes variable-level dependencies among slots. Two slots are considered dependent if they are occupied by the same variable or by variables constrained by an inequality. Based on this dependency analysis, the algorithm partitions all slots into disjoint independent subsets. It then transforms each *setP* construct in the assertion into a tensor product of smaller ones according to the partition. This is the first key technique for cost reduction.

This strategy works best when slots are mutually independent, reducing construction complexity from exponential to linear in the number of variables. In the following example, since the two slots on the left-hand side are independent, this phase transforms the expression as follows:

$$\left\{ \sum_{|i|=5,\ |j|=5} |ij\rangle \right\} \longrightarrow \left\{ \sum_{|i|=5} |i\rangle \right\} \otimes \left\{ \sum_{|j|=5} |j\rangle \right\}$$

This allows subsequent steps to process i and j separately, reducing the enumeration count from $2^{|i|+|j|}$ (1024 states) to $2^{|i|} + 2^{|j|}$ ($32 + 32 = 64$ states). Conversely, such a reduction is unachievable when all slots are mutually dependent. In either case, the complexity is guaranteed not to increase.

Slot Reordering. We model the slot dependencies using an undirected graph $G = (\mathcal{K}, E)$, where $\mathcal{K}$ consists of slot indices. An edge $(i, j) \in E$ (with $i \neq j$) exists if and only if there exist variables u at slot i and v at slot j that satisfy the **recurrence condition** (i.e., refer to the same variable instance) or the **inequality condition** (i.e., are constrained by the inequality $u \neq v$). After building this graph, we compute the connected components. To ensure

a deterministic transformation structure, we first arrange these components in ascending order of their minimum slot indices. Then, we arrange the vertices in each component into a list and sort each list in ascending order. Finally, we obtain a new total slot order by concatenating these ordered lists.

Example 4 (Slot Reordering). Consider two *setP* constructs S_A and S_B within a tensor segment, sharing the same 7-slot structure.

$$S_A = \left\{ \alpha_{1A} \sum_{\substack{|a|=1,\ |b|=1,\\ |c|=2,\ |d|=2,\\ e=0,\ a\neq b}} |abcxwde\rangle + \alpha_{2A} \sum_{\substack{|f|=1,\ |g|=1,\\ |h|=2,\ |j|=2,\\ |k|=1}} |fghijwk\rangle : \begin{matrix} |i|=1,\\ |w|=2,\\ |x|=1 \end{matrix} \right\},$$

$$S_B = \left\{ \alpha_{1B} \sum_{\substack{|l|=1,\ |q|=2,\\ |m|=2,\ |n|=1,\\ y=0,\ p\neq q}} |lupyqmn\rangle + \alpha_{2B} \sum_{\substack{|o|=1,\ |r|=1,\\ |s|=2,\ |t|=1,\\ |v|=2}} |orstzvu\rangle : \begin{matrix} |p|=2,\\ |u|=1,\\ |z|=2,\\ p\neq z \end{matrix} \right\}.$$

We construct the dependency graph for slot indices $\{1, 2, \ldots, 7\}$. In S_A, the recurrence condition arises from variable w occupying slot 5 in the first term and slot 6 in the second term. This creates edge $(5, 6)$. The inequality condition arises from $a \neq b$ in the first term, which creates edge $(1, 2)$. In S_B, the recurrence condition arises from variable u occupying slot 2 in the first term and slot 7 in the second term. This creates edge $(2, 7)$. The inequality condition arises from $p \neq z$ in the set predicate and $p \neq q$ in the first term, which both create edge $(3, 5)$. The resulting dependency graph and its connected components are illustrated in Figure 5. The resulting new total slot order is therefore $\big[[1, 2, 7], [3, 5, 6], [4]\big]$.

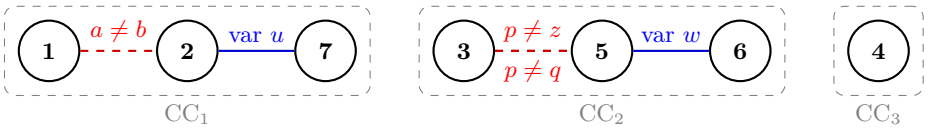


Fig. 5. Dependency graph for slot indices based on S_A and S_B. Solid blue lines indicate recurrence dependencies (shared variables), and dashed red lines indicate inequality constraints. The graph reveals three disjoint connected components. In each component, the numbers are arranged in ascending order. In the whole graph, the components are arranged in ascending order of their minimum elements.

Tensor Product Transformation. Assume a *setP* construct S occupies the variable slots reordered by concatenating k lists $L_1, \ldots, L_k$, each corresponding to a connected component from slot reordering. Based on this new slot order, we transform S into a tensor product structure $S_1 \otimes S_2 \otimes \cdots \otimes S_k$, where each S_i is obtained by projecting S onto the slots in L_i. For clarity, we designate these S_i

components as *setV* constructs (extending the syntax in Figure 2) to explicitly identify them as intermediate outcomes of the variable-level transformation. To facilitate this transformation and prevent undesired recombinations of projected terms[4] originating from distinct source *term* constructs during the final composition, we introduce *tag amplitudes* $\mathcal{T} = \{\tau_0, \ \ldots, \ \tau_t\}$[5], where t is the number of *term* constructs in S. Here, τ_0 is the additive identity and is absorbing for multiplication. These tags satisfy the idempotence and orthogonality properties:

$$\tau_m \cdot \tau_n = \begin{cases} \tau_m & \text{if } m = n, \\ \tau_0 & \text{otherwise.} \end{cases}$$

With tag amplitudes, we construct each *setV* construct S_i as follows.

1. **VStr Extraction:** For each *term* construct, we derive a new string *VStr*2 from *VStr* by selecting the variables located at the slot indices in the order specified by L_i, and then replace the original *VStr* with this new *VStr*2.
2. **Constraint Filtering:** Regardless of whether constraints appear under the summation or within the set predicate, we retain only those involving the variables present in *VStr*2. Irrelevant constraints are discarded accordingly.
3. **Amplitude Replacement:** The amplitude α_m of the m-th *term* is replaced by the tag τ_m. This replacement uniquely tags the term, ensuring only the projected terms originating from the same source term are recombined during the tensor product.

The resulting set S_i comprises t instances of a variant construct, denoted as *termV*. Extending the syntax in Figure 2, *termV* is structurally identical to *term*, except that the complex amplitude is replaced by a tag amplitude.

Example 5 (Tensor Product Transformation). Recall in Example 4 that the resulting new total slot order is $\big[[1,2,7],[3,5,6],[4]\big]$ and both sets (S_A and S_B) are transformed according to this order. For instance, S_B is transformed into $S_{B,1} \otimes S_{B,2} \otimes S_{B,3}$:

$$S_{B,1} = \left\{ \tau_1 \sum_{\substack{|l|=1,\\ |n|=1}} |lun\rangle + \tau_2 \sum_{\substack{|o|=1,\\ |r|=1}} |oru\rangle : |u| = 1 \right\},$$

$$S_{B,2} = \left\{ \tau_1 \sum_{\substack{|q|=2,\\ |m|=2,\\ p\neq q}} |pqm\rangle + \tau_2 \sum_{\substack{|s|=2,\\ |v|=2}} |szv\rangle : \begin{matrix} |p|=2, \\ |z|=2, \\ p\neq z \end{matrix} \right\}, \ S_{B,3} = \left\{ \tau_1 \sum_{y=0} |y\rangle + \tau_2 \sum_{|t|=1} |t\rangle \right\}.$$

[4] Projected terms are of the form $\tau_m \, |VStr\rangle$ or $\tau_m \sum_{varcons} \, |VStr\rangle$.

[5] The domain $\{\sum_{m\in T} \tau_m\}_{T\in 2^{\mathcal{T}}\setminus\{\tau_0\}\setminus\{\emptyset\}} \cup \{\tau_0\}$, equipped with the defined binary operations, forms a commutative nonunital semiring.

5.4 Qubit-Level Reordering and Tensor Product Transformation

Unlike the preceding variable-level transformation, which treats variables as atomic units, inequalities such as $i \neq j$ can always be resolved at the qubit level via the logical disjunction of $i_k \neq j_k$ across all qubits k (i.e., $i_1 \neq j_1 \vee \ldots \vee i_\ell \neq j_\ell$). This observation enables us to further decompose the dependency structure: distinct qubits can be handled in separate constructs, linked only by the accumulated satisfaction status of constraints. By expanding variables into individual qubits, this phase effectively reduces the time complexity from exponential to linear in the number of qubits.

Let ℓ denote the qubit length of the variables in a *setV* construct S derived from the previous phase. We expand each multi-qubit variable v into a sequence of single-qubit variables $v_1 v_2 \ldots v_\ell$. Based on this expansion, we transform S into a qubit-level tensor product structure $S_1 \otimes S_2 \otimes \cdots \otimes S_\ell$, where each S_k governs the k-th qubit slice. We refer to each component S_k as a *setQ* construct (an internal structure not covered in Figure 2), which utilizes a specialized amplitude form to track the partial satisfaction of constraints (e.g., $i \neq j \iff \bigvee_k (i_k \neq j_k)$).

Valuation-Dependent Amplitudes. To implement disjunctive logic while maintaining the accumulated satisfaction status, we introduce the concept of *valuation-dependent amplitudes*. Consider a *setV* construct containing t instances of *termV*. For the m-th *termV*, let Φ_m denote the set of inequality constraints collected from both the local summation within *termV* and the global set predicate of *setV*. We define the valuation-dependent amplitude as a collection of boolean functions. Formally, let $d \triangleq \{f_m\}_{m \in T}$ be a set for some subset of term indices $T \subseteq [t]$, where each element $f_m : \Phi_m \to \mathbb{B}$ maps the constraints in Φ_m to truth values[6].

The algebraic operations on $d_1 \triangleq \{f_m^1\}_{m \in T_1}$ and $d_2 \triangleq \{f_m^2\}_{m \in T_2}$ are defined as follows:

- **Sum:** $d_1 + d_2 \triangleq \{f_m\}_{m \in T_1 \cup T_2}$, where $f_m \triangleq f_m^1$ if $m \in T_1$ and f_m^2 if $m \in T_2$. It is well-defined because in this work, this operation is applied only when $T_1 \cap T_2 = \emptyset$.
- **Product:** $d_1 \cdot d_2 \triangleq \{f_m\}_{m \in T_1 \cap T_2}$, where $f_m(\phi) \triangleq f_m^1(\phi) \vee f_m^2(\phi), \quad \forall \phi \in \Phi_m$.

With these definitions, we construct each *setQ* construct S_k from S as follows:

1. **Qubit Projection and Constraint Relaxation:** We project all variables in S onto their k-th qubits (adding subscript k). Initially, we relax all constraints, meaning that all single-qubit variables are freely instantiated to values in $\{0, 1\}$.

[6] The amplitude domain consists of all partial mappings $d = \{f_m\}_{m \in T}$ from subsets of term indices $T \subseteq [t]$ to boolean functions. By extending the sum operation for overlapping indices $m \in T_1 \cap T_2$ as the pointwise disjunction $f_m(\phi) \triangleq f_m^1(\phi) \vee f_m^2(\phi)$, this domain, equipped with the defined binary operations, forms a commutative nonunital semiring.

2. **Constraint Evaluation:** The previously disregarded constraints are then integrated into the valuation-dependent amplitudes. Specifically, the original tag amplitude τ_m is replaced by the singleton set $\{f_m\}$, where the truth value of each constraint in Φ_m is determined locally by the current assignment of the qubit-level variables.
3. **Concrete Expansion:** Finally, each quantum state in *setQ* is expanded into a summation of concrete basis states with these newly computed valuation-dependent amplitudes. These concrete expansions are then passed to the automaton construction procedure described in the next section.

Example 6 (Qubit-Level Construction for $S_{B,2}$ in Example 5). Recall that

$$S_{B,2} = \left\{ \tau_1 \sum_{\substack{|q|=2,\\ |m|=2,\\ p\neq q}} |pqm\rangle + \tau_2 \sum_{\substack{|s|=2,\\ |v|=2}} |szv\rangle : \begin{smallmatrix} |p|=2,\\ |z|=2,\\ p\neq z \end{smallmatrix} \right\}.$$

Since the qubit length is 2, we construct two qubit slices $S^{(1)}_{B,2}$ and $S^{(2)}_{B,2}$. The compact algebraic form[7] of the j-th qubit slice ($j = 1, 2$) is given by:

$$S^{(j)}_{B,2} = \left\{ \sum_{q_j,m_j} \left\{ f_1^{V_1} \right\} |p_j q_j m_j\rangle + \sum_{s_j,v_j} \left\{ f_2^{V_2} \right\} |s_j z_j v_j\rangle : p_j, z_j \right\},$$

where the constraint sets are $\Phi_1 \triangleq \{p \neq z,\ p \neq q\}$ and $\Phi_2 \triangleq \{p \neq z\}$. For brevity, we denote the valuation functions as $f_m^{V_m}$, where V_1 and V_2 represent the boolean assignments to the variables $\{p_j, z_j, q_j, m_j\}$ and $\{p_j, z_j, s_j, v_j\}$, respectively. The function bodies are determined locally: $f_1^{V_1}(p \neq z) \triangleq (p_j \neq z_j)$, $f_1^{V_1}(p \neq q) \triangleq (p_j \neq q_j)$, and $f_2^{V_2}(p \neq z) \triangleq (p_j \neq z_j)$.

5.5 Compact LSTA Construction

For a set of quantum states in Dirac notation, we construct an LSTA for each quantum state and take the set union of the resulting LSTAs via the LSTA set union operation. Recalling that a quantum state can be represented by a perfect binary tree, we construct the LSTA directly mimicking this tree structure, with all transitions enabled by the singleton choice $\{1\}$. This compact construction holds for any valid amplitude domain $\mathbb{K}$ and yields an upper bound guarantee on the resulting automaton size by employing a bottom-up approach that merges isomorphic subtrees. The result is summarized in the following theorem (see Appendix A.2 in [34] for the complete proof).

Theorem 7. *Let $|\psi\rangle = \sum_{s\in\{0,1\}^n} a_s |s\rangle$ be an n-qubit state. Let $N = |\ \{s \in \{0,1\}^n \mid a_s \neq 0_{\mathbb{K}}\}\ |$ denote the number of nonzero-amplitude terms. The size of the LSTA $\mathcal{A} = \langle Q, V, \Delta, r\rangle_{\mathbb{K}}$, constructed via the levelwise procedure detailed below, is bounded by $|\Delta| = O(N \cdot n)$.*

[7] The concrete expansion is given in Example 8.

The construction for a quantum state in the form of $\sum_{s\in\{0,1\}^n \,:\, a_s\neq 0_{\mathbb{K}}} a_s\,|s\rangle$ proceeds level by level, from the leaves up to the root:

(1) **Leaf Level:** We create a state q_s for each basis state $|s\rangle$ following a nonzero coefficient $a_s \in \mathbb{K}$, assigning the leaf transition $q_s \xrightarrow{\{1\}} a_s$. Additionally, we construct a default sink state $q_\perp^n$ with $q_\perp^n \xrightarrow{\{1\}} 0_{\mathbb{K}}$ to handle missing terms. Let Q_n denote the set of these explicit states.
(2) **Internal Levels (Iterate ℓ from $n-1$ down to 0):** At each level, we first construct a sink state $q_\perp^\ell$ with the internal transition $q_\perp^\ell \xrightarrow{\{1\}} (q_\perp^{\ell+1}, q_\perp^{\ell+1})$. Next, identifying the set of active prefixes $X_\ell = \{x \in \{0,1\}^\ell \mid q_{x0} \in Q_{\ell+1} \vee q_{x1} \in Q_{\ell+1}\}$, we construct a state q_x for each $x \in X_\ell$ with the transition $q_x \xrightarrow{\{1\}} (u_0, u_1)$. Here, the child state u_b (for $b \in \{0,1\}$) is resolved to the explicit state q_{xb} if $q_{xb} \in Q_{\ell+1}$, and defaults to $q_\perp^{\ell+1}$ otherwise. The set Q_ℓ is then updated to include these newly created states.
(3) **Root Assignment:** The state $q_\epsilon \in Q_0$ is designated as the root state.

It is worth noting that if $a_s \neq 0_{\mathbb{K}}$ for all $s \in \{0,1\}^n$ (i.e., full support), the construction of sink states and their associated transitions becomes superfluous and can be skipped.

Example 8 (Concrete Expansion and LSTA Construction for $S_{B,2}^{(j)}$ in Ex. 6).

$$\text{Recall that } S_{B,2}^{(j)} = \left\{ \sum_{q_j,\ m_j} \left\{f_1^{V_1}\right\} |p_j q_j m_j\rangle + \sum_{s_j,\ v_j} \left\{f_2^{V_2}\right\} |s_j z_j v_j\rangle : p_j, z_j \right\} =$$

$$\underbrace{\left\{ \begin{array}{l} \left\{f_1^{\{p\neq z\mapsto \mathsf{F},\ p\neq q\mapsto \mathsf{F}\}}\right\}(|000\rangle + |001\rangle) \\ + \left\{f_1^{\{p\neq z\mapsto \mathsf{F},\ p\neq q\mapsto \mathsf{T}\}}\right\}(|010\rangle + |011\rangle) \\ + \left\{f_2^{\{p\neq z\mapsto \mathsf{F}\}}\right\}(|000\rangle + |001\rangle + |100\rangle + |101\rangle) \end{array} \right\}}_{p_j=0,\ z_j=0} \cup \underbrace{\left\{ \begin{array}{l} \left\{f_1^{\{p\neq z\mapsto \mathsf{T},\ p\neq q\mapsto \mathsf{F}\}}\right\}(|000\rangle + |001\rangle) \\ + \left\{f_1^{\{p\neq z\mapsto \mathsf{T},\ p\neq q\mapsto \mathsf{T}\}}\right\}(|010\rangle + |011\rangle) \\ + \left\{f_2^{\{p\neq z\mapsto \mathsf{T}\}}\right\}(|010\rangle + |011\rangle + |110\rangle + |111\rangle) \end{array} \right\}}_{p_j=0,\ z_j=1}$$

$$\cup \underbrace{\left\{ \begin{array}{l} \left\{f_1^{\{p\neq z\mapsto \mathsf{T},\ p\neq q\mapsto \mathsf{T}\}}\right\}(|100\rangle + |101\rangle) \\ + \left\{f_1^{\{p\neq z\mapsto \mathsf{T},\ p\neq q\mapsto \mathsf{F}\}}\right\}(|110\rangle + |111\rangle) \\ + \left\{f_2^{\{p\neq z\mapsto \mathsf{T}\}}\right\}(|000\rangle + |001\rangle + |100\rangle + |101\rangle) \end{array} \right\}}_{p_j=1,\ z_j=0} \cup \underbrace{\left\{ \begin{array}{l} \left\{f_1^{\{p\neq z\mapsto \mathsf{F},\ p\neq q\mapsto \mathsf{T}\}}\right\}(|100\rangle + |101\rangle) \\ + \left\{f_1^{\{p\neq z\mapsto \mathsf{F},\ p\neq q\mapsto \mathsf{F}\}}\right\}(|110\rangle + |111\rangle) \\ + \left\{f_2^{\{p\neq z\mapsto \mathsf{F}\}}\right\}(|010\rangle + |011\rangle + |110\rangle + |111\rangle) \end{array} \right\}}_{p_j=1,\ z_j=1}.$$

In the above expansion, the superscripts of functions f_m are changed from V_m to the definition body of f_m, implying that the explicit valuations of single-qubit variables are no longer required for the subsequent steps. In each case $p_j \in \{0,1\}$, $z_j \in \{0,1\}$, we construct the LSTA for the corresponding quantum state. We then take the union of these four LSTAs to obtain the final LSTA for $S_{B,2}^{(j)}$. Detailed transition sets are provided in the full version [34].

5.6 Final Assembly of LSTA

Up to this point, all *set* constructs within the assertions have been broken down into *setQ* constructs with their corresponding qubit-level LSTAs $\{\mathbf{M}_Q\}$. First, these LSTAs are combined via the tensor product operation to form variable-level LSTAs $\{\mathbf{M}'_V\}$, following the specified decomposition structure. To filter out invalid assignments, we apply a filter mapping filter_f to the valuation-dependent amplitudes in each $\mathbf{M}'_V$. We denote $f_m \equiv \mathbf{1}$ if $f_m(\phi) = \mathsf{T}$ for all $\phi \in \Phi_m$. Accordingly, the mapping is defined as $\mathsf{filter}_f(\{f_m\}_{m\in T}) = \sum_{m\in T, f_m\equiv\mathbf{1}} \tau_m$ (retaining terms where all applicable constraints are satisfied), or τ_0 if the set $\{m \in T : f_m \equiv \mathbf{1}\}$ is empty. This effectively replaces leaf transitions $q \xrightarrow{C} e$ with $q \xrightarrow{C} \mathsf{filter}_f(e)$, resulting in LSTAs $\{\mathbf{M}_V\}$ that represent *setV* constructs. This mapping serves as a *validation filter*, ensuring that each $\mathbf{M}_V$ encapsulates only valid *termV* constructs.

Example 9 (Application of filter_f *to Amplitudes in Ex. 8).* The mapping operates on Δ as follows: $\mathsf{filter}_f\left(\left\{f_2^{\{p\neq z\mapsto \mathsf{T}\}}\right\}\right) = \tau_2$, $\mathsf{filter}_f\left(\left\{f_1^{\left\{\substack{p\neq z\mapsto \mathsf{T}\\ p\neq q\mapsto \mathsf{T}}\right\}},\ f_2^{\{p\neq z\mapsto \mathsf{T}\}}\right\}\right)$ $= \tau_1 + \tau_2$, $\mathsf{filter}_f\left(\left\{f_1^{\left\{\substack{p\neq z\mapsto \mathsf{T}\\ p\neq q\mapsto \mathsf{F}}\right\}}\right\}\right) = \tau_0$, and $\mathsf{filter}_f(\emptyset) = \tau_0$.

Remark. This is only for demonstration. The real application will occur right after constructing $\bigotimes_j \mathrm{LSTA}(S^{(j)}_{B,2}) \in \{\mathbf{M}'_V\}$ to obtain $\mathrm{LSTA}(S_{B,2}) \in \{\mathbf{M}_V\}$.

Subsequently, the LSTAs in $\{\mathbf{M}_V\}$ are again tensored according to the higher-level decomposition structure to produce LSTAs $\{\mathbf{M}'_P\}$. We apply another filter mapping filter_τ, defined by $\mathsf{filter}_\tau(\sum_{m\in T}\tau_m) = \sum_{m\in T\setminus\{0\}}\alpha_m$ and 0 if $T = \{0\}$, to the amplitudes in each $\mathbf{M}'_P$. This transforms leaf transitions $q \xrightarrow{C} e$ into $q \xrightarrow{C} \mathsf{filter}_\tau(e)$, yielding the final LSTAs $\{\mathbf{M}_P\}$ for *setP* constructs. This second stage ensures only projected terms originating from the same source term are recomposed during the tensor product.

Finally, all LSTAs in $\{\mathbf{M}_P\}$ now possess standard amplitudes in $\mathbb{C}[V_c]$. They are further fused by successively applying LSTA set union and tensor product operations as indicated by the remaining operators in the assertion to obtain the final LSTA $\mathcal{M}$.

As such, each assertion reduces to the form $\bigcup_{\theta\models CCons}\mathcal{L}(\mathcal{M}(\theta))$, where θ is a valuation of complex variables. This formulation signifies that the assertion represents a set of concrete quantum states, each derived from a symbolic tree recognized by $\mathcal{M}$ under a concrete instantiation satisfying the constraint *CCons*.

5.7 Space Complexity of LSTA

In the following theorem[8], we establish the space complexity of the LSTA constructed from an assertion. We characterize the complexity in terms of the total

[8] See Appendix A.2 in [34] for the complete proof.

number of qubits (L) and four auxiliary structural parameters: the number of *term* constructs (N_{term}) and *varcon* constructs (N_{vc}) within each *set* construct, the number of *dirac* constructs (N_{union}) within each *uset* construct, and the total number of distinct symbolic amplitudes (N_{amp}) in the assertion.

Theorem 10. *The size of the final LSTA $\mathcal{M}$ constructed from an assertion is bounded by* $|\Delta| = O\left(2^{N_{term} \cdot 2^{N_{vc}}} \cdot 2^{N_{vc}} \cdot N_{term} \cdot N_{union} \cdot L \cdot N_{amp}\right)$.

Our complexity is *linear* in the key parameter L. This efficiency stems from transforming assertions into tensor product structures, leveraging the additive complexity of LSTA tensor product operations with a scaling multiplier. The remaining structural parameters are independent of L and negligible in practice; in our experiments, all such parameters are bounded by 2 (see Table 2 in Appendix A.3 in [34]).

6 Experimental Results

We implemented the specification-to-automata translation framework proposed in this work and compared it with the translation algorithm in the latest version of AutoQ [12][9]. The latter uses the same specification language as described in [13], but translates specifications into LSTAs.

We evaluated performance on a suite of representative circuits: Bernstein-Vazirani (BV), Greenberger-Horne-Zeilinger state preparation [21] (GHZ), the multi-controlled Toffoli gate (MCToffoli), Grover's search (Grover), and one iteration of Grover's search (GroverIter), as detailed in Section 4. These benchmarks encompass canonical quantum algorithms and standard composite gates. The preconditions and postconditions for these benchmarks are listed in Table 2 in Appendix A.3 in [34].

For the oracle-based algorithms (BV, Grover, and GroverIter), we verified their parameterized-oracle versions to guarantee correctness for all possible oracles. This was achieved by using the first n qubits to control the X gates in the oracle via CX gates. For GHZ, we input all basis states to demonstrate the bitwise complement feature of our language. For MCToffoli, we verified the four specific cases indicated in Section 4.3 individually to ensure the functional correctness of the implementation.

We conducted all experiments on a server running Ubuntu 24.04.3 LTS, equipped with an AMD EPYC 7742 64-core processor (1.5 GHz), 2 TiB of RAM, and a 4 TB SSD. A timeout of 5 minutes was enforced for each circuit verification. The results are presented in Table 1. The proposed translation algorithm is consistently faster than the original AutoQ 2.0 algorithm. Moreover, it produces smaller LSTAs, leading to shorter verification times. These results highlight the scalability achieved by our approach.

[9] https://github.com/fmlab-iis/AutoQ

Table 1. Results of verifying our use cases with this work and [12]. Columns #**q** and #**G** denote the number of qubits and gates of the circuit, respectively. For each case, we report the time required to: translate the specification into precondition and post-condition LSTAs (trans), perform the verification process (ver), and the total running time (total). The timeout is 5 minutes.

	#q	#G	This Work			AutoQ [12]		
			trans	ver	total	trans	ver	total
BV	17	27	0.0s	0.0s	**0.0s**	1.5s	0.9s	**2.4s**
	19	30	0.0s	0.0s	**0.0s**	6.1s	4.3s	**10.4s**
	21	33	0.0s	0.0s	**0.0s**	25s	19.2s	**44.2s**
	23	36	0.0s	0.0s	**0.0s**	2m19s	1m35s	**3m54s**
	25	39	0.0s	0.0s	**0.0s**	TIMEOUT		
GHZ	8	8	0.0s	0.0s	**0.0s**	0.6s	0.2s	**0.8s**
	9	9	0.0s	0.0s	**0.0s**	2.7s	1.1s	**3.8s**
	10	10	0.0s	0.0s	**0.0s**	11s	4.8s	**15.8s**
	11	11	0.0s	0.0s	**0.0s**	44s	19.1s	**1m3s**
	12	12	0.0s	0.0s	**0.0s**	TIMEOUT		
Grover	20	544	0.0s	0.2s	**0.2s**	0.5s	2.9s	**3.4s**
	23	927	0.0s	0.5s	**0.5s**	1.9s	10.1s	**12s**
	26	1475	0.0s	1s	**1s**	7.3s	42s	**49.3s**
	29	2408	0.0s	1.9s	**1.9s**	30s	3m33s	**4m3s**
	32	3711	0.0s	3.5s	**3.5s**	TIMEOUT		

	#q	#G	This Work			AutoQ [12]		
			trans	ver	total	trans	ver	total
GroverIter	20	79	0.0s	0.1s	**0.1s**	0.8s	2.7s	**3.5s**
	23	91	0.0s	0.1s	**0.1s**	2.9s	8.3s	**11.2s**
	26	103	0.0s	0.1s	**0.1s**	11s	30.7s	**41.7s**
	29	115	0.0s	0.1s	**0.1s**	50s	2m19s	**3m9s**
	32	127	0.0s	0.1s	**0.1s**	TIMEOUT		
* MCToffoli	16	15	0.0s	0.0s	**0.0s**	1.2s	0.4s	**1.6s**
	18	17	0.0s	0.0s	**0.0s**	4.4s	1.6s	**6s**
	20	19	0.0s	0.0s	**0.0s**	18.1s	7.3s	**25.4s**
	22	21	0.0s	0.0s	**0.0s**	1m13s	30.4s	**1m43s**
	24	23	0.0s	0.0s	**0.0s**	TIMEOUT		

*) Running time in this benchmark is the sum of four cases indicated in Section 4.3.

Cross-Paradigm Comparison. Beyond the performance improvements in translation complexity, the tree-automata-based paradigm utilized in this work has been extensively benchmarked against various automatic verification tools in prior studies [1, 12–15]. These include symbolic verifiers such as symQV [5] (based on the SMT theory of reals) and CAAL [16] (based on an extended SMT theory of arrays), as well as simulators like the state-vector-based SV-Sim [26] and the decision-diagram-based SliQSim [35]. Furthermore, the paradigm has been compared with the Feynman tool suite [3] (based on the path-sum) and Qcec [9] (which integrates decision diagrams, the ZX-calculus [17], and random stimuli generation [8]). These evaluations demonstrate that while primarily focusing on pure-state families, the set-based, automata-driven approach outperforms these counterparts in terms of running time and scalability, particularly for large-scale instances. We omit direct comparisons with Hermitian-based or projection-based tools as they typically prioritize the breadth of mixed-state logic and often require non-trivial manual proof developments in interactive theorem provers, whereas our approach provides a fully automated, "push-button" verification path for large-scale circuits.

7 Concluding Remarks

We bridged the gap between expressive high-level specifications and fully automated quantum program verification. By identifying the exponential blow-up in prior automata-based approaches as a primary bottleneck, we introduced an extended specification language capable of describing complex families of quantum

states and proposed a novel translation algorithm. By leveraging variable-level and qubit-level reordering strategies to facilitate tensor product decomposition, our algorithm reduces the construction complexity of LSTAs from exponential to linear in the number of qubits. Our experimental results demonstrate that this approach dramatically improves scalability, enabling the verification of large-scale circuits that were previously beyond the reach of existing automated tools.

To better situate our framework within the landscape of quantum verification, it is important to distinguish its underlying philosophy from traditional methods. Our approach adopts an extensional view of specifications, where predicates are identified with sets of quantum states and represented using a compact automata-based structure. Although this differs from the intensional, formula-based approach in Hoare-style logics, it still supports principled reasoning through set-theoretic operations such as inclusion, union, and intersection, reflecting a different balance between expressiveness and tractability. An important property of LSTAs is that they are not closed under complement. Supporting arbitrary negation would require leaving the automata domain, thereby sacrificing compactness and automation. We therefore treat this as a deliberate restriction to preserve tractability. However, we retain a meaningful notion of implication through set inclusion, which can be efficiently checked within our automata framework. Such restrictions are not entirely foreign to logical systems. For instance, in intuitionistic logic, negation is not treated as a primitive operator in the same way as in classical logic, and reasoning often proceeds via implication. Overall, we view our approach as complementary to QHL-style frameworks: logical specifications provide generality and strong reasoning principles, while our representation-aligned design enables scalable and fully automatic verification in a practically important regime.

Looking ahead, there are several promising avenues for extending this framework. One potential direction involves the automatic splitting of atomic variables to facilitate variable alignment, which is particularly beneficial when manual decomposition is complex. Such an extension could be realized by enhancing *varcon* constructs to support logical disjunction. Additionally, the framework could be further extended to support classical-quantum states, enabling the verification of programs with mid-circuit measurements and sophisticated classical control flow. This might be achieved by, for instance, employing equalities to bind classical and quantum states to variables, offering a more flexible representation. These developments would further enhance the practical utility and automated capabilities of our framework, ultimately paving the way for verifying realistic hybrid quantum-classical workflows.

Acknowledgments. We thank the reviewers for their constructive feedback. This work was supported by National Science and Technology Council, R.O.C., project NSTC 114-2119-M-001-002; Air Force Office of Scientific Research project FA2386-23-1-4107; Academia Sinica Investigator Project Grant AS-IV-114-M07; the Czech Science Foundation project 25-18318S; and the FIT BUT internal project FIT-S-26-9011.

Disclosure of Interests. The authors have no relevant financial or non-financial interests to disclose.

Data Availability Statement. The code and benchmarks for reproducing the findings of this paper are available on Zenodo at `https://doi.org/10.5281/zenodo.19724316` [33].

References

1. Abdulla, P.A., Chen, Y., Chen, Y., Holík, L., Lengál, O., Lin, J., Lo, F., Tsai, W.: Verifying quantum circuits with level-synchronized tree automata. Proc. ACM Program. Lang. **9**(POPL), 923–953 (2025). `https://doi.org/10.1145/3704868`
2. Abdulla, P.A., Chen, Y.F., Hečko, M., Holík, L., Lengál, O., Lin, J.A., Thinniyam, R.S.: Parameterized verification of quantum circuits. Proc. ACM Program. Lang. **10**(POPL) (Jan 2026). `https://doi.org/10.1145/3776712`
3. Amy, M.: Towards large-scale functional verification of universal quantum circuits. In: Selinger, P., Chiribella, G. (eds.) Proceedings 15th International Conference on Quantum Physics and Logic, QPL 2018, Halifax, Canada, 3-7th June 2018. EPTCS, vol. 287, pp. 1–21 (2018). `https://doi.org/10.4204/EPTCS.287.1`
4. Arute, F., Arya, K., Babbush, R., et al.: Quantum supremacy using a programmable superconducting processor. Nature **574**, 505–510 (2019). `https://doi.org/10.1038/s41586-019-1666-5`
5. Bauer-Marquart, F., Leue, S., Schilling, C.: symqv: Automated symbolic verification of quantum programs. In: Chechik, M., Katoen, J., Leucker, M. (eds.) Formal Methods - 25th International Symposium, FM 2023, Lübeck, Germany, March 6-10, 2023, Proceedings. pp. 181–198. Lecture Notes in Computer Science, Springer (2023). `https://doi.org/10.1007/978-3-031-27481-7_12`
6. 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`
7. Birkhoff, G., von Neumann, J.: The logic of quantum mechanics. Annals of Mathematics **37**(4), 823–843 (1936). `https://doi.org/10.2307/1968621`
8. Burgholzer, L., Kueng, R., Wille, R.: Random stimuli generation for the verification of quantum circuits. In: ASPDAC '21: 26th Asia and South Pacific Design Automation Conference, Tokyo, Japan, January 18-21, 2021. pp. 767–772. ACM (2021). `https://doi.org/10.1145/3394885.3431590`
9. Burgholzer, L., Wille, R.: Advanced equivalence checking for quantum circuits. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **40**(9), 1810–1824 (2021). `https://doi.org/10.1109/TCAD.2020.3032630`
10. Chareton, C., Bardin, S., Bobot, F., Perrelle, V., Valiron, B.: An automated deductive verification framework for circuit-building quantum programs. In: Yoshida, N. (ed.) Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12648, pp. 148–177. Springer (2021). `https://doi.org/10.1007/978-3-030-72019-3_6`
11. Chen, T., Chen, Y., Jiang, J.R., Jobranová, S., Lengál, O.: Accelerating quantum circuit simulation with symbolic execution and loop summarization. In: Xiong, J., Wille, R. (eds.) Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2024, Newark Liberty International Airport Marriott, NJ, USA, October 27-31, 2024. pp. 42:1–42:9. ACM (2024). `https://doi.org/10.1145/3676536.3676711`

12. Chen, Y., Chung, K., Hsieh, M., Huang, W., Lengál, O., Lin, J., Tsai, W.: Autoq 2.0: From verification of quantum circuits to verification of quantum programs. In: Gurfinkel, A., Heule, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 31st International Conference, TACAS 2025, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, ON, Canada, May 3-8, 2025, Proceedings, Part III. Lecture Notes in Computer Science, vol. 15698, pp. 87–108. Springer (2025). https://doi.org/10.1007/978-3-031-90660-2_5
13. Chen, Y., Chung, K., Lengál, O., Lin, J., Tsai, W.: Autoq: An automata-based quantum circuit verifier. In: Enea, C., Lal, A. (eds.) Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13966, pp. 139–153. Springer (2023). https://doi.org/10.1007/978-3-031-37709-9_7
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. Proc. ACM Program. Lang. **7**(PLDI), 1218–1243 (2023). https://doi.org/10.1145/3591270
15. Chen, Y., Chung, K., Lengál, O., Lin, J., Tsai, W., Yen, D.: An automata-based framework for verification and bug hunting in quantum circuits. Commun. ACM **68**(6), 85–93 (2025). https://doi.org/10.1145/3725728
16. Chen, Y., Rümmer, P., Tsai, W.: A theory of cartesian arrays (with applications in quantum circuit verification). In: Pientka, B., Tinelli, C. (eds.) Automated Deduction - CADE 29 - 29th International Conference on Automated Deduction, Rome, Italy, July 1-4, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14132, pp. 170–189. Springer (2023). https://doi.org/10.1007/978-3-031-38499-8_10
17. Coecke, B., Duncan, R.: Interacting quantum observables: categorical algebra and diagrammatics. New Journal of Physics **13**(4), 043016 (2011). https://doi.org/10.1088/1367-2630/13/4/043016
18. D'Hondt, E., Panangaden, P.: Quantum weakest preconditions. Math. Struct. Comput. Sci. **16**(3), 429–451 (2006). https://doi.org/10.1017/S0960129506005251
19. Feng, Y., Ying, M.: Quantum hoare logic with classical variables. ACM Trans. Quantum Comput. **2**(4), 16:1–16:43 (2021). https://doi.org/10.1145/3456877
20. Feng, Y., Zhou, L., Xu, Y., Xu, X.: Refinement calculus of quantum programs with projective assertions. ACM Trans. Softw. Eng. Methodol. (Oct 2025). https://doi.org/10.1145/3770083, just Accepted
21. Greenberger, D.M., Horne, M.A., Zeilinger, A.: Going beyond Bell's theorem. In: Kafatos, M. (ed.) Bell's Theorem, Quantum Theory and Conceptions of the Universe, pp. 69–72. Springer Netherlands, Dordrecht (1989). https://doi.org/10.1007/978-94-017-0849-4_10
22. 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
23. Harrigan, M.P., Sung, K.J., Neeley, M., et al.: Quantum approximate optimization of non-planar graph problems on a planar superconducting processor. Nature Physics **17**, 332–336 (2021). https://doi.org/10.1038/s41567-020-01105-y
24. Kim, Y., et al.: Evidence for the utility of quantum computing before fault tolerance. Nature **618**, 500–505 (2023). https://doi.org/10.1038/s41586-023-06096-3
25. Lewis, M., Zuliani, P., Soudjani, S.: Automated verification of Silq quantum programs using SMT solvers. In: Chang, R.N., Chang, C.K., Yang, J., Jin, Z.,

Sheng, M., Fan, J., Fletcher, K., He, Q., Faro, I., Leymann, F., Barzen, J., de la Puente, S., Feld, S., Wimmer, M., Atukorala, N., Wu, H., Elkouss, D., García-Alonso, J., Sarkar, A. (eds.) IEEE International Conference on Quantum Software, QSW 2024, Shenzhen, China, July 7-13, 2024. pp. 125–134. IEEE (2024). https://doi.org/10.1109/QSW62656.2024.00027

26. Li, A., Fang, B., Granade, C.E., Prawiroatmodjo, G., Heim, B., Roetteler, M., Krishnamoorthy, S.: Sv-sim: scalable pgas-based state vector simulation of quantum circuits. In: de Supinski, B.R., Hall, M.W., Gamblin, T. (eds.) International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021. p. 97. ACM (2021). https://doi.org/10.1145/3458817.3476169
27. Li, L., Zhu, M., Cleaveland, R., Nicolellis, A., Lee, Y., Chang, L., Wu, X.: Qafny: A quantum-program verifier. In: Aldrich, J., Salvaneschi, G. (eds.) 38th European Conference on Object-Oriented Programming, ECOOP 2024, Vienna, Austria, September 16-20, 2024. LIPIcs, vol. 313, pp. 24:1–24:31. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024). https://doi.org/10.4230/LIPICS.ECOOP.2024.24
28. 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: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11562, pp. 187–207. Springer (2019). https://doi.org/10.1007/978-3-030-25543-5_12
29. 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, November 20-22, 1994. pp. 124–134. IEEE Computer Society (1994). https://doi.org/10.1109/SFCS.1994.365700
30. Singhal, K., Reppy, J.H.: Quantum hoare type theory: Extended abstract. In: Valiron, B., Mansfield, S., Arrighi, P., Panangaden, P. (eds.) Proceedings 17th International Conference on Quantum Physics and Logic, QPL 2020, Paris, France, June 2 - 6, 2020. EPTCS, vol. 340, pp. 291–302 (2020). https://doi.org/10.4204/EPTCS.340.15
31. Stamatopoulos, N., Egger, D.J., Sun, Y., Zoufal, C., Iten, R., Shen, N., Woerner, S.: Option pricing using quantum computers. Quantum **4**, 291 (2020). https://doi.org/10.22331/Q-2020-07-06-291
32. Sundaram, A., Rand, R., Singhal, K., Lackey, B.: Hoare meets Heisenberg: A lightweight logic for quantum programs (2025), https://arxiv.org/abs/2101.08939
33. Tsai, W.L., Chen, Y.F., Lengál, O.: A Practical Specification Language for Automatic Quantum Program Verification. Zenodo. https://doi.org/10.5281/zenodo.19724316
34. Tsai, W.L., Chen, Y.F., Lengál, O.: A practical specification language for automatic quantum program verification (technical report) (2026), https://arxiv.org/abs/2605.05786
35. 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
36. Unruh, D.: Quantum hoare logic with ghost variables. In: 34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada,

June 24-27, 2019. pp. 1–13. IEEE (2019). https://doi.org/10.1109/LICS.2019.8785779
37. Yan, P., Jiang, H., Yu, N.: On incorrectness logic for quantum programs. Proc. ACM Program. Lang. **6**(OOPSLA1), 1–28 (2022). https://doi.org/10.1145/3527316
38. Yan, P., Jiang, H., Yu, N.: Approximate relational reasoning for quantum programs. In: Gurfinkel, A., Ganesh, V. (eds.) Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part III. Lecture Notes in Computer Science, vol. 14683, pp. 495–519. Springer (2024). https://doi.org/10.1007/978-3-031-65633-0_22
39. Ying, M.: Floyd-Hoare logic for quantum programs. ACM Trans. Program. Lang. Syst. **33**(6), 19:1–19:49 (2011). https://doi.org/10.1145/2049706.2049708
40. Younes, A.: Strength and Weakness in Grover's Quantum Search Algorithm (2008), https://arxiv.org/abs/0811.4481
41. Yu, N., Palsberg, J., Reps, T.: A logic for approximate quantitative reasoning about quantum circuits (2025), https://arxiv.org/abs/2507.13635
42. Zhou, L., Barthe, G., Strub, P., Liu, J., Ying, M.: Coqq: Foundational verification of quantum programs. Proc. ACM Program. Lang. **7**(POPL), 833–865 (2023). https://doi.org/10.1145/3571222
43. Zhou, L., Yu, N., Ying, M.: An applied quantum hoare logic. In: McKinley, K.S., Fisher, K. (eds.) Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019. pp. 1149–1162. ACM (2019). https://doi.org/10.1145/3314221.3314584