On the Verification of Quantum Circuits (Research Challenges and Opportunities)

Parosh Aziz Abdulla[1,2], Yo-Ga Chen[3], Yu-Fang Chen[3], Lukáš Holík[6], Ondřej Lengál[4], Jyun-Ao Lin[5], Fang-Yi[3], and Lo Wei-Lun Tsai

[1] Uppsala University, Sweden
[2] Mälardalen University, Sweden
[3] Academia Sinica, Taiwan
[4] Brno University of Technology, Czechia
[5] National Taipei University of Technology, Taiwan
[6] Aalborg University, Denmark

Abstract. Quantum technology is advancing at an exceptional pace and holds the potential to transform numerous sectors on both national and global scales.
As quantum systems become more sophisticated and widespread, ensuring their correctness becomes critically important. This highlights the pressing need for rigorous tools capable of analyzing and verifying their behavior.
However, developing such verification tools poses significant challenges. Fundamental quantum phenomena—most notably superposition and entanglement—lead to program behaviors that differ radically from those in classical computing. These characteristics give rise to inherently probabilistic models and result in exponentially large state spaces, even for systems of modest complexity.
In this paper, we outline initial steps toward addressing these challenges by drawing on insights gained from the verification of classical systems within our community.
We then present a roadmap for designing novel verification frameworks that adapt the strengths of classical methods—such as succinct property specification, precise fault detection, automation, and scalability—to the quantum setting.

1 Introduction

Classical computing has advanced considerably over the decades and now forms the backbone of modern technological infrastructure. In contrast, quantum computing introduces a fundamentally new paradigm for information processing, grounded in the principles of quantum mechanics. This paradigm enables the solution of computational problems that are beyond the practical capabilities of classical methods.

The core distinction between classical and quantum computing lies in the way information is represented and manipulated. Classical computers operate on bits that exist in one of two binary states—0 or 1. In contrast, quantum

computers use quantum bits, or *qubits*, which exhibit the quantum phenomena of *superposition* and *entanglement*. Superposition enables a qubit to exist in multiple states at once, thereby allowing parallel evaluation of many possibilities. Entanglement creates correlations between qubits such that the state of one qubit can instantaneously influence the state of another, regardless of the distance between them. These phenomena enable quantum systems to represent and manipulate exponentially many states simultaneously, and to encode complex correlations that classical systems cannot efficiently reproduce, resulting in powerful computational advantages.

000	0	100	0
001	0	101	$\frac{1}{3}$
010	0	110	$\frac{1}{3}$
011	$\frac{\sqrt{6}}{3}$	111	$\frac{1}{3}$

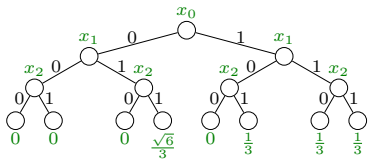


Fig. 1: A quantum state with three qubits and its tree representation. The system can be in any of the eight states, with the probability given in the table.

Fig. 1 depicts a quantum state consisting of three qubits.[7] In a classical circuit, the system would occupy a single *basis* state from among the eight possible configurations: 000, 001, ..., 111. By contrast, a quantum system can exist in a superposition of all eight basis states simultaneously, each associated with a specific probability amplitude.

A quantum state can be interpreted as a distribution over basis states, where each state is assigned a complex amplitude. This distribution may be visualized as a tree, in which each path from the root to a leaf corresponds to a basis state, and each leaf stores a complex-valued amplitude. Unlike classical probabilistic models, quantum amplitudes are complex numbers, and the sum of the squares of their absolute values must equal one.[8]

A primary objective in quantum computing is to achieve *quantum supremacy*—the point at which quantum computers can solve problems that are infeasible for even the most powerful classical supercomputers. Reaching this milestone would result in exponential speedups for specific computational tasks.

[7] Throughout the text, we illustrate the challenges encountered in verifying quantum circuits through examples. For clarity, these examples are simplified. Full technical details are beyond the scope of this document and can be found, for example, in [Yanofsky and Mannucci, 2008].

[8] An *amplitude* generalizes the classical notion of probability. The square of the absolute value of a complex amplitude gives the corresponding probability. The use of complex numbers permits constructive and destructive interference, allowing for the cancellation of certain outcomes.

Although the field is still in an early phase of practical realization, quantum computing applications are growing rapidly. In the near term, hybrid systems integrating classical and quantum processors are expected to gain prominence. Looking ahead, quantum supremacy could revolutionize entire industries by solving problems that were previously considered intractable.

Potential application areas for quantum technologies include medical diagnostics, cryptography, secure communications infrastructure, and autonomous systems, among others. These domains typically require stringent reliability and safety standards, as they cannot tolerate critical system failures. Consequently, they demand rigorous certification procedures and robust assurance mechanisms. Ensuring the reliability of quantum systems is therefore of critical importance.

Nevertheless, verifying quantum systems presents formidable challenges. Their probabilistic behavior, combined with the exponential growth of state spaces as the number of qubits increases, introduces substantial complexity.

In this paper, we describe initial steps toward tackling these challenges by leveraging insights from our community's extensive experience in verifying classical systems. Building on this foundation, we will also propose a roadmap for the development of new verification frameworks that transfer the proven strengths of classical approaches—such as concise property specification, accurate fault detection, automation, and scalability—into the quantum domain.

We envision significant added value in bridging two complementary areas of computer science, especially when established techniques from a mature field are adapted to address complex problems in an emerging domain. This work embodies such interdisciplinary integration by applying well-developed methods from logic, automata theory, and symbolic verification to the problem of ensuring the correctness of quantum programs.

We present a novel application of automata theory—a foundational discipline in formal verification—to the verification of quantum systems. Our methodology integrates automata-based reasoning with symbolic representations tailored for the automated analysis of quantum behavior. The goal is to generalize core concepts from classical verification, such as state-space exploration and symbolic reasoning, into the quantum realm, thereby enabling the adaptation of robust classical paradigms to quantum computing, where formal assurances are essential.

Interestingly, this approach also opens new avenues for research in automata theory itself, as the mathematical structures encountered in quantum systems introduce novel challenges and opportunities. It establishes a conceptual bridge between quantum program verification and automata theory, fostering potential for new theoretical developments and extending the expressive and analytical power of automata-based techniques into the context of quantum computation.

We will give a high-level description of how to develop *symbolic representations*, based on *automata*, that serve as a theoretical and algorithmic foundation for efficiently modeling the state spaces of quantum circuits. These representations are grounded in the idea of modeling quantum states as binary trees, where

each path from the root to a leaf corresponds to a computational basis state (see Fig. 1).

In the remainder of the paper, we first describe the modeling of quantum states, gates, and circuits, followed by a formalization of the verification problem. Subsequently, we outline a roadmap highlighting potential challenges and opportunities in the verification of quantum systems.

2 States

We present the two standard representations of quantum states used in quantum computing: the matrix (vector) representation and the Dirac notation. We then introduce our tree-based representation. We begin with the special case of classical states.

In classical computing, bits are the fundamental units of information. Each bit can exist in one of two possible states: 0 or 1. These may be conceptualized as switches that are either turned on or off. In contrast, quantum computing employs quantum bits, or *qubits*, to represent and process information. Qubits can exist in a *superposition*, meaning they may simultaneously be in state 0, state 1, or a combination of both. Additionally, a qubit can be subjected to a *measurement* operation. The state of a qubit is described by two complex numbers, called *amplitudes*, c_0 and c_1, where $|c_0|^2$ denotes the probability of measuring the state as 0, and $|c_1|^2$ the probability of measuring it as 1. These amplitudes must satisfy the normalization condition: $|c_0|^2 + |c_1|^2 = 1$.

$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$	$\begin{bmatrix} \sqrt{1/2} \\ \sqrt{1/2} \end{bmatrix}$	$\begin{bmatrix} 1/3 \\ \sqrt{8}/3 \end{bmatrix}$	$\begin{bmatrix} (2-i)/3 \\ 2i/3 \end{bmatrix}$
$\|0\rangle$	$\|1\rangle$	$\sqrt{1/2}\,\|0\rangle + \sqrt{1/2}\,\|1\rangle$	$1/3\,\|0\rangle + \sqrt{8}/3\,\|1\rangle$	$(2-i)/3\,\|0\rangle + 2i/3\,\|1\rangle$
(a)	(b)	(c)	(d)	(e)

Fig. 2: Different qubits and their vector and Dirac representations. (a, b) The basis states $|0\rangle$ and $|1\rangle$. (c, d, e) Quantum states in superposition. Measuring the state in (c) yields 0 or 1 with probability 0.5. Measuring the state in (d) yields 0 with probability 1/9 and 1 with probability 8/9.

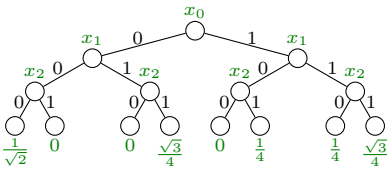
There are two common representations used to describe a qubit state (cf. Fig. 2). The first is the vector notation, in which the state is written as a column vector of length 2, with complex entries corresponding to the amplitudes of the respective basis states. To simplify notation, we often use the transpose of this column vector. For example, we write $[1/3 \quad \sqrt{8}/3]^T$ to represent the state shown in Fig. 2 (d). This results in a more compact row vector form. The second

representation is the Dirac notation, where the state is expressed as a linear combination, e.g., $|0\rangle + |1\rangle$. Classical states are special cases where either $c_0 = 1$ and $c_1 = 0$, or $c_0 = 0$ and $c_1 = 1$. In quantum systems, these are referred to as the basis states $|0\rangle$ and $|1\rangle$.

We now generalize these concepts to systems with multiple qubits. For n qubits, there are 2^n possible basis states. In the case of two qubits, the system has four basis states. These can be represented as column vectors:

$$[1\ 0\ 0\ 0]^\intercal \quad [0\ 1\ 0\ 0]^\intercal \quad [0\ 0\ 1\ 0]^\intercal \quad [0\ 0\ 0\ 1]^\intercal$$

$$\left[\tfrac{1}{\sqrt{2}}\ \ 0\ \ 0\ \ \tfrac{\sqrt{3}}{4}\ \ 0\ \ \tfrac{1}{4}\ \ \tfrac{1}{4}\ \ \tfrac{\sqrt{3}}{4}\right]^T$$



$$(1/\sqrt{2})\,|000\rangle + (\sqrt{3}/4)\,|011\rangle + (1/4)\,|101\rangle + (1/4)\,|110\rangle + (\sqrt{3}/4)\,|111\rangle$$

Fig. 3: A state involving three qubits. We provide the vector, Dirac, and tree-based representations. The tree spans all basis states, with the corresponding amplitudes shown as leaf labels. For instance, the left-most path represents $|000\rangle$ with amplitude $1/\sqrt{2}$.

A key feature of our framework is a symbolic representation of quantum states and sets of quantum states. We utilize *tree automata* as the underlying symbolic structure. Specifically, a quantum state over n qubits is encoded as a perfect binary tree of depth n. Each level of the tree corresponds to one qubit. We refer to the level associated with qubit x as the x-level, and the nodes at that level as x-nodes. For any x-node, its left and right subtrees represent the cases where qubit x has values 0 and 1, respectively. Each root-to-leaf path encodes a computational basis state. The leaf nodes hold the complex amplitudes associated with those basis states.

Consider Fig. 1. The left side shows a quantum state involving three qubits. The right side displays its corresponding tree representation.

3 Gates

We describe gate behaviors using matrices, and then describe the effect of a gate as a transformation of the tree representing the input state. We start by modeling classical digital gates in the quantum setting. Then, we introduce some quantum gates and show we can represent them using matrices.

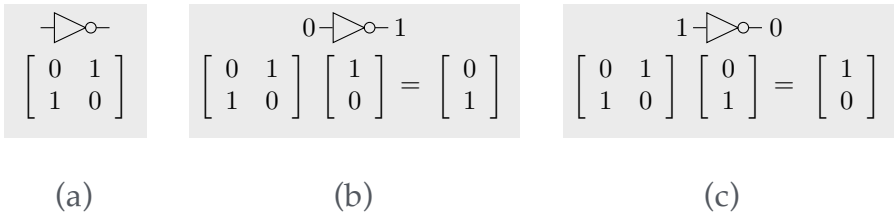


Fig. 4: (a) The `NOT` gate together with its matrix presentation. (b) Applying the gate to 0, represented by $[1 \;\; 0]^T$, will give 1, i.e., $[0 \;\; 1]^T$. (c) Applying the gate to 1 will give 0.

Classical Gates Logical gates, such as `AND`, `OR`, and `NOT`, are the fundamental components of digital circuits. They take binary inputs (0s and 1s) and produce a single binary output. Each type of gate implements a specific binary function, e.g., the output of an `AND` gate is the conjunction of its inputs. In order to prepare for quantum gates, we will represent the behavior of classical gates by matrices. Let us take the `NOT` gate as an example (Fig. 4). For a given (basis) state, we compute the result of applying the gate by multiplying the matrix representing the gate with the input state. The gate has one input and one output bit, and hence its behavior is described by a 2×2-matrix. The input and output states are both column vectors.

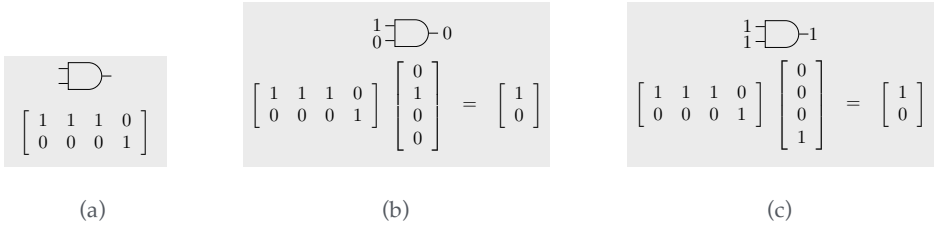


Fig. 5: (a) The `AND gate` together with its matrix presentation. (b) Applying the gate to 01, represented by $[0 \;\; 1 \;\; 0 \;\; 0]^T$, will give 0. (c) Applying the gate to 11, represented by $[0 \;\; 0 \;\; 0 \;\; 1]^T$, will give 1.

Fig. 5 depicts the `AND` gate. The gate has two input bits and one output bit, and hence its behavior is described by a 2×4-matrix. The input state is of length 4 and the output state has length 2.

Quantum Gates In a similar manner to classical gates, we will represent quantum gate behaviors by matrices. Quantum gates are *reversible*: if we run the gate and then run the inverse of the gate, then we get the identity matrix. Formally,

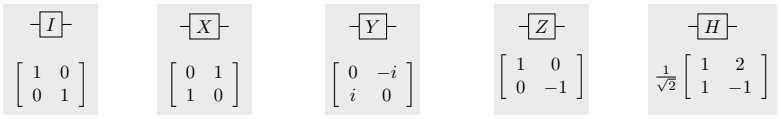


Fig. 6: The Identity gate, the Pauli gates, and the Hadamard gate.

the matrix representing the gate should be *unitary*, i.e., its conjugate transpose should be equal to its inverse. In Fig. 6 we show some examples of quantum gates and their behaviors. The gate `I` is the identity, and maps each input to an identical output. Next, we introduce the three Pauli gates.

The $\texttt{Pauli}-\texttt{X}$ is the quantum analogue of the classical `NOT` gate, acting on a single qubit. It is often referred to as *bit-flip gate* because it inverts the state. In classical computing, a `NOT` gate flips a binary value—changing 0 to 1 and vice versa. The $\texttt{Pauli}-\texttt{X}$ gate performs the same operation in the quantum domain for the basis states.

$$X|0\rangle = |1\rangle \qquad X|1\rangle = |0\rangle$$

Moreover, unlike classical bits, qubits can exist in *superpositions* of $|0\rangle$ and $|1\rangle$. The $\texttt{Pauli}-\texttt{X}$ gate also acts on such states by swapping the amplitudes of the two basis states. For example, consider a qubit in the superposition:

$$|\psi\rangle = a|0\rangle + b|1\rangle$$

where a and b are complex probability amplitudes. Applying the $\texttt{Pauli}-\texttt{X}$ gate yields:

$$X|\psi\rangle = b|0\rangle + a|1\rangle$$

Thus, the $\texttt{Pauli}-\texttt{X}$ gate exchanges the amplitudes of $|0\rangle$ and $|1\rangle$, effectively flipping the qubit's state within the superposition.

The $\texttt{Pauli}-\texttt{Z}$ gate is also known as the phase-flip gate. Unlike the $\texttt{Pauli}-\texttt{X}$ gate—which acts as a quantum analogue of the classical `NOT` gate by flipping the state between $|0\rangle$ and $|1\rangle$, the $\texttt{Pauli}-\texttt{Z}$ gate leaves the computational basis states unchanged but inverts the phase of the $|1\rangle$ component. In other words, it modifies the relative phase between $|0\rangle$ and $|1\rangle$, a core concept in quantum mechanics. Thus, applying the $\texttt{Pauli}-\texttt{Z}$ gate to a qubit in state $|0\rangle$ has no effect, while applying it to $|1\rangle$ multiplies the state by -1, effectively flipping its phase. Due to this nature, $\texttt{Pauli}-\texttt{Z}$ is sometimes called phase-flip. This operation is essential in many quantum algorithms and plays a critical role in phase manipulation and interference, which are central to quantum computation.

The $\texttt{Pauli}-\texttt{Y}$ gate is a combination of the $\texttt{Pauli}-\texttt{X}$ and $\texttt{Pauli}-\texttt{Z}$ gates, meaning it applies both a bit-flip and a phase-flip to the qubit. It changes both the state of the qubit (like the $\texttt{Pauli}-\texttt{X}$ gate), and the relative phase between the states $|0\rangle$ and $|1\rangle$ (like the $\texttt{Pauli}-\texttt{Z}$ gate).

For example, if you have a qubit in the state $|0\rangle$ and apply the $\texttt{Pauli}-\texttt{Y}$ gate, the qubit's state changes to the state $i|1\rangle$. If you have a qubit in the state $|1\rangle$ and

apply the $\texttt{Pauli} - \texttt{Y}$ gate, the qubit's state changes to the state $-i|0\rangle$. Consider a qubit in the following superposition state: $|\psi\rangle = a|0\rangle + b|1\rangle$, where a and b are complex numbers representing the amplitudes of the qubit states. When the $\texttt{Pauli} - \texttt{Y}$ gate is applied to this qubit, the result will be: $Y|\psi\rangle = -ib|0\rangle + ia|1\rangle$. We observe that the $\texttt{Pauli} - \texttt{Y}$ gate has not only swapped the amplitudes of the $|0\rangle$ and $|1\rangle$ states but also added a relative phase of i to the resulting states.

The *Hadamard* gate (often denoted as $\texttt{H}$) is a fundamental single-qubit gate in quantum computing. It creates superposition states, which are essential for quantum parallelism. It transforms the basis states as follows:

$$\texttt{H}|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \quad \texttt{H}|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

The $\texttt{H}$ gate has three important properties:

- *Self-inverse*: $\texttt{H}^2 = \texttt{I}$, so applying it twice returns the original state.
- Creates superposition: Crucial for quantum algorithms like Deutsch-Jozsa, Grover's, and Shor's.
- Balanced output: Equal amplitude for both $|0\rangle$ and $|1\rangle$ (up to a sign), enabling interference.

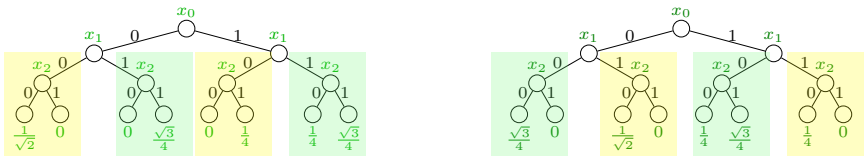


Fig. 7: Applying the X gate on an input tree. We apply the gate to the bit x_1. The application is modeled by swapping the left and right subtrees of all nodes at the level corresponding to the qubit x_1 (i.e., level 1 in this case).

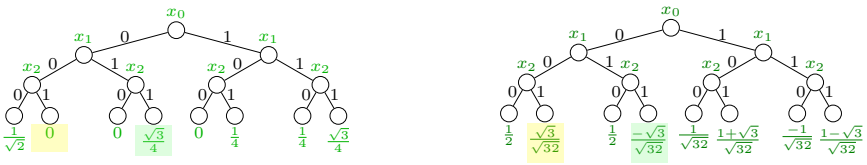


Fig. 8: Applying the $\texttt{H}$ gate on an input tree. The gate is applied to the bit x_1. This transformation also targets all nodes at level 1 (corresponding to x_1), updating subtree structure and amplitudes accordingly.

Tree Transformations In our setting, we encode gate applications as tree transformations. Given a tree representing a quantum state, a qubit x, and a gate, we compute a new tree that describes the quantum state resulting from applying the gate to the given qubit.

A major difference from the classical setting is that, due to quantum superposition, the effect of a gate application is generally *global*. That is, we must update all parts of the state in which the qubit x is involved.

In Fig. 7 and Fig. 8, we illustrate this by swapping all subtrees at nodes corresponding to x_1. As shown in the example, a gate application can affect the amplitudes of exponentially many leaves—effectively updating an exponential number of classical states.

4 Circuits

A combinatorial circuit consists of a set of gates and acts as a Boolean function. It takes a sequence of bits as input and produces a sequence of bits as output. There are also internal bits that represent the connections between gates. In the figure, the circuit has three input bits, one output bit, and three internal bits.

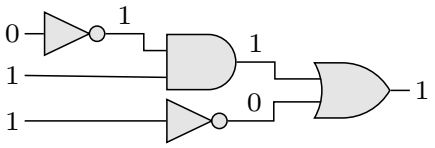


Fig. 9: A classical combinatorial circuit

Fig. 10 shows a quantum circuit consisting of an `H` followed by a $\texttt{Pauli} - \texttt{Z}$ gate.

We start with a qubit in a superposition state created using a Hadamard gate, and then apply a $\texttt{Pauli} - \texttt{Z}$ gate to it.

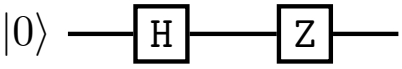


Fig. 10: A simple quantum circuit.

Here is a step-by-step description of the circuit behavior when the starting qubit is $|0\rangle$.

1. Start in $|0\rangle = [1 \;\; 0]^T$
2. Apply the Hadamard gate: $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
   Now the qubit is in an equal superposition.
3. Apply the $\texttt{Pauli} - \texttt{Z}$ gate: $\texttt{Pauli} - \texttt{Z}\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right) = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.
   The $|1\rangle$ term's sign flips—this is phase inversion.

Intuitively, the Hadamard spreads the probability amplitude across both states.

The $\texttt{Pauli} - \texttt{Z}$ keeps the probability unchanged (as it only affects the phase), but introduces a phase shift to $|1\rangle$, which is crucial in interference patterns for quantum algorithms.

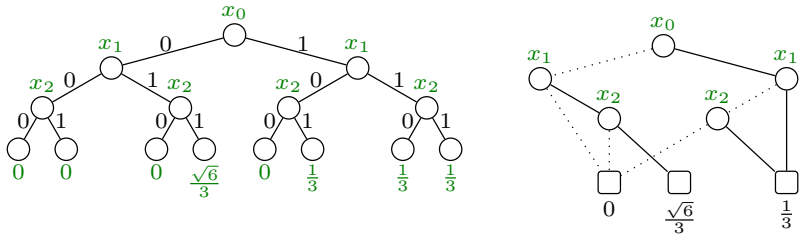


Fig. 11: The BDD representation of a quantum state.

Next, we turn our attention to the verification of quantum circuits.

We can formulate the *program verification* problem as an instance of classical Hoare Logic:

$$P\{C\}Q$$

where P is the *pre-condition*, Q is the post-condition, and C is a program.

The pre-condition P characterizes a set of *initial* (*source*) states, and the post-condition Q characterizes a set of *final* (*target*) states.

We would like to verify that if we execute C from any state satisfying P, then we end up in a state satisfying Q.

Here, we take C to be a quantum circuit.

A fundamental breakthrough in the verification of conventional computer systems was the invention of efficient data structures to represent sets of states.

A case in point is the classical BDD (Binary Decision Diagram) data structure.

Fig. 11 depicts a BDD representation of a quantum state.

The main challenge in using BDDs in quantum circuits is the fact that a BDD represents a single state.

In conventional circuits, a BDD represents a (large) set of states.

Thus, we need a framework in which we can handle sets of BDDs.

Given that we represent quantum states by trees, a natural choice is to use tree automata to represent sets of quantum states.

Fig. 12(a) gives a tree automaton for accepting all the basis states of size three.

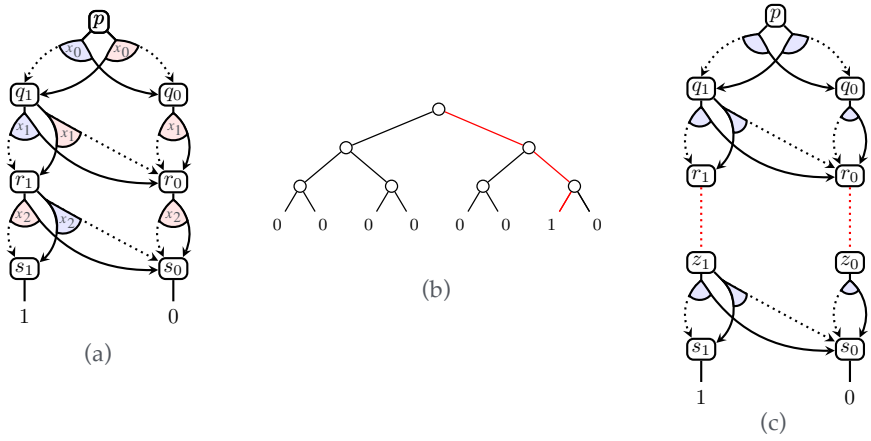


Fig. 12: (a) A tree automaton for generating all basis states of size 3. The dashed lines represent the left child, and the bold lines represent the right child. (b) The tree corresponding to $[0\ 0\ 0\ 0\ 0\ 0\ 1\ 0]^T$. The red rules in the automaton are those used to generate the tree. (c) An automaton to generate all basis states of size n.

The set of rules is:

$$
\begin{array}{lll}
q \xrightarrow{x_0} (q_0, q_1) & q \xrightarrow{x_0} (q_1, q_0) & \\
q_0 \xrightarrow{x_1} (r_0, r_0) & q_1 \xrightarrow{x_1} (r_1, r_0) & q_1 \xrightarrow{x_1} (r_0, r_1) \\
r_0 \xrightarrow{x_2} (s_0, r_0) & r_1 \xrightarrow{x_2} (s_1, s_0) & r_1 \xrightarrow{x_2} (s_0, s_1) \\
s_0 \to 0 & s_1 \to 1 &
\end{array}
$$

Notice that we are characterizing 2^n basis states using only $3n+1$ transitions.

In our setting, a gate application corresponds to a tree transformation.

For a circuit consisting of a number of gates, we provide an algorithm that transforms a tree automaton describing a set of input states to a new automaton describing the set of output states.

We do this by constructing a sequence of automata that model the effects of each gate.

Fig. 13 gives a concrete example to demonstrate our approach.

Assume we want to design a circuit constructing the Bell state, i.e., a 2-qubit circuit converting a basis state $|00\rangle$ to the maximally entangled state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

Given the automaton corresponding to the pre-condition (Fig. 13(a)), we derive the automaton corresponding to the post-condition (Fig. 1b).

In this case, both automata use q as the root state and accept only one tree.

We can observe the correspondence between quantum states and tree automata by traversing their structures.

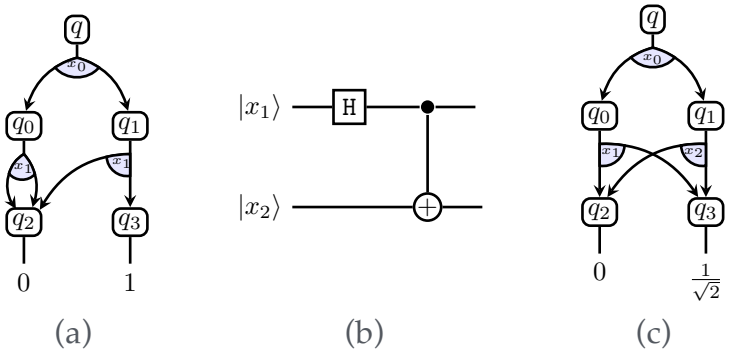


Fig. 13: The Bell gates

5 Challenges

Research on the algorithmic verification of quantum systems remains in its early stages. Only recently have initial contributions emerged, including the first studies employing SAT solvers for quantum verification [Chen et al., 2023b]. We have taken foundational steps in this direction through recent work on the symbolic verification of quantum circuits [Abdulla et al., 2025b,Abdulla, 2025,Abdulla et al., 2025a].

In contrast, the algorithmic verification of classical systems, particularly through model checking [Abdulla et al., 2018], has been an active and well-established area of research for nearly four decades. This field has yielded a substantial and mature body of work and continues to thrive, with ongoing research published regularly in premier venues such as POPL, CAV, PLDI, MICRO, ASPLOS, etc.

While extending verification techniques to quantum systems is a natural and necessary progression, it also introduces a host of novel and technically challenging problems. In the remainder of this section, we briefly review the current state of the art and its limitations.

5.1 Symbolic Encodings

Symbolic state space encodings have been among the most influential techniques in the verification of conventional programs over the past three decades. By representing program semantics symbolically using various data structures and applying mathematical reasoning, one can efficiently verify whether a program satisfies its correctness properties. These data structures encode logical formulas, allowing for both compact modeling of large or infinite state spaces and the efficient execution of logical operations. Notable examples of symbolic representations that have significantly advanced the verification of classical systems include Binary Decision Diagrams (BDDs) for hardware verification and communication protocols, as well as automata over words and trees, for verifying parameterized systems and programs with dynamic heap structures.

Despite their success in conventional computing, the application of symbolic techniques remains limited in the quantum domain. This limitation stems from quantum computation's fundamentally different mathematical foundations, which involve state superposition, probabilistic measurement, and non-local entanglement—features that challenge conventional symbolic methods. Nevertheless, with techniques likely to be developed in the future, symbolic verification will eventually play an equally important role in advancing the scalability and precision of quantum program verification.

As quantum computing continues to evolve, there is a growing need to develop symbolic abstractions tailored to quantum systems, hybrid symbolic-numeric approaches, and novel verification frameworks. These directions represent promising and largely unexplored areas of research.

The limitations of the state of the art include the following:

- With the exception of [Chen et al., 2023a] and our own recent contributions [Abdulla et al., 2025b,Abdulla et al., 2025a], there is currently a lack of work dedicated to developing symbolic encodings tailored for the automated verification of quantum circuits. By contrast, several studies have focused on adapting classical Hoare logic to quantum systems. In particular, the approach proposed in [D'Hondt and Panangaden, 2006] and Ying's quantum Hoare logic [Ying, 2012] defines predicates as mappings from mixed states—i.e., probability distributions over pure quantum states—to real values in the interval $[0, 1]$, representing the probability that a given state satisfies a particular condition. However, these logics are inherently deductive and interactive, not algorithmic in nature. As such, they are not suited for automated verification, often requiring significant manual effort and domain-specific reasoning.
  An alternative is to adopt a fundamentally different approach by proposing a set-based methodology, where predicates are defined as mappings from quantum states to the Boolean domain $\{0, 1\}$. Within this framework, automata serve as compact representations of such predicates. A tree representing a quantum state is accepted by an automaton if and only if the corresponding predicate maps that state to 1. One of the primary advantages of this set-based formulation is that it facilitates the construction of efficient and fully automated verification algorithms, making it significantly more scalable for practical use in the quantum domain.
- Currently, there are no algorithms for minimization, simulation, and bisimulation over automata models designed specifically for the verification of quantum systems. Addressing this gap should be a central objective. We need to adapt and extend prior experience in classical automata theory to develop such algorithmic frameworks tailored to the specific characteristics of quantum systems.

5.2 Algorithmic Verification

As previously mentioned, algorithmic verification techniques such as *model checking* have been among the most influential methodologies for checking the correct-

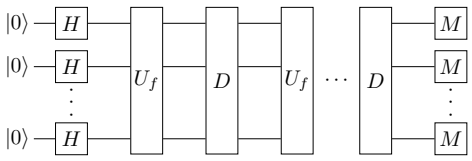


Fig. 14: The classical Grover's search algorithm.

ness of classical programs over the past three decades. They offer an automated mechanism to determine whether a system satisfies a given specification. Crucially, model checking is a complete verification method, meaning it can guarantee the absence of errors—unlike testing, which can only reveal the presence of bugs. Nevertheless, extending model checking techniques to quantum systems presents considerable challenges due to three primary reasons.

First, as previously discussed, quantum systems involve state spaces of infinite size, even when the system comprises only a finite number of qubits.

Second, quantum systems are intrinsically parameterized, typically along two dimensions. Consider, for instance, the circuit illustrated in Fig. 14, which implements the classical Grover's algorithm. This circuit accepts a Boolean function with n input variables and aims to find an input for which the function evaluates to 1. If such an input exists, the circuit identifies one with a runtime of $O(\sqrt{N})$, where $N = 2^n$. In contrast, any classical algorithm addressing the same problem would require $O(N)$ time. Since the algorithm is constructed to operate on an arbitrary number of input qubits, there is no fixed upper bound on the input size. Moreover, the system architecture comprises several stages, determined by $O(\sqrt{N})$. Intuitively, each algorithm stage carries out one so-called *Grover rotation*. This is an operation that increases the amplitudes of the basis states that satisfy the input function f. By performing an increasing number of iterations, the precision of the computation increases, and the probability that we will measure the correct answer approaches one. We reach the required precision after $O(\sqrt{N})$ iterations, which explains the number of stages[9]. The system is characterized by a two-dimensional parameterization: the number of qubits and the number of stages are unbounded. We need to verify the system's correctness across the entire space of possible values for these two parameters.

Third, unlike classical systems, quantum systems exhibit globally entangled transitions. In general, the application of a gate to a single qubit may affect an exponential number of classical basis states. For instance, in Fig. 1, negating the qubit x_1 would swap the leaves in positions 1, 3, 5, and 7 with the leaves in positions 2, 4, 6, and 8, respectively.

The following aspects are currently lacking in existing work:

– Thge paper [Bauer-Marquart et al., 2023] employs *symbolic execution* [King, 1976] to verify input-output relationships by formulating queries discharged to

[9] Again, we give a high-level description here. We refer, e.g., to [Yanofsky and Mannucci, 2008] for the algorithm details.

SMT solvers over the theory of real numbers. The SMT array theory approach [Chen et al., 2023b] instantiated this framework by enabling a polynomial-sized encoding of quantum circuits. Nevertheless, it continues to suffer from scalability limitations. In [Abdulla et al., 2025b,Abdulla et al., 2025a], we demonstrate that automata-based techniques outperform these approaches by several orders of magnitude. Another scalable, fully automated method for analyzing quantum circuits is *quantum abstract interpretation* [Yu and Palsberg, 2021,Perdrix, 2008]. However, quantum abstract interpretation relies on over-approximation, which may limit its precision in bug detection.

– There are no methods supporting regular model checking or parameterized verification for quantum applications. Our recent paper [Abdulla et al., 2025b] presents two examples of parameterized quantum circuits. However, it does not provide algorithms for general parameterized verification. We need to bridge this gap by designing and integrating model checking algorithms with abstraction techniques tailored for quantum systems.

6 Research Opportunities

We outline some opportunities for research in quantum circuit verification.

6.1 Automata

We need to provide a comprehensive road map for designing efficient symbolic encodings based on automata to represent the state spaces of quantum systems. Furthermore, we should develop the first algorithms for minimization and checking language inclusion, language equivalence, simulation, and bisimulation relations for such automata.

Language Inclusion and Equivalence A central challenge in formal verification, in general, and for quantum systems in particular, is *language inclusion*. We can use the operation to ensure that a system's behavior conforms to its specification and to formulate termination conditions within verification procedures. However, in many cases, checking language inclusion is computationally intensive or even undecidable. This problem is well known in classical automata and similarly manifests in the models for quantum systems in [Abdulla et al., 2025b]. Therefore, we anticipate the models introduced in this project will exhibit the same computational difficulty. To address this, we may employ *simulation* relations as an efficient alternative and enhance them with subset-construction-like techniques to improve precision. We expect these simulation-based methods to be significantly more efficient than direct language inclusion checks when applied to quantum constraints. The disadvantage of using simulation relations is their incompleteness: a simulation preorder implies language inclusion, but the converse does not necessarily hold. To deal with incompleteness, we can, in parallel, develop methods based on the classical subset-construction for checking language inclusion. As in the classical case, we expect these algorithms to

suffer from state space explosion, with an exponential increase in the number of states. To combine the strengths of both approaches, we need to adapt the antichain framework introduced in [Wulf et al., 2006,Abdulla et al., 2010]. In this setting, the computed simulation relation will be used to prune unnecessary search paths during the execution of the subset-construction-based algorithm, thereby improving both performance and scalability.

Minimization It is vital to design and implement algorithms to minimize the automata models developed in the project. Minimization techniques are essential for ensuring the scalability of our tools, as automata will serve as symbolic representations of state spaces. Consequently, the size of these automata directly impacts the computational complexity of language inclusion, simulation, and bisimulation problems addressed in Section 6.1, which underpin the verification procedures described in Section 6.2.

In general, computing a minimal automaton requires a determinization step, which may cause an intermediate exponential blow-up in the automaton's size. As a result, despite the potential compactness of the final minimized automaton, the computational cost of determinization may render the approach impractical, constituting a bottleneck in our verification pipeline.

We can adopt a pragmatic and computationally feasible strategy to circumvent this challenge: merge automaton states according to simulation and bisimulation relations. Although these relations are stricter than language equivalence, they can be computed efficiently (as discussed above). This yields a practical trade-off between the efficiency of the minimization algorithm and the degree of reduction it achieves.

6.2 Verification Techniques

We need to develop algorithms for parameterized verification of quantum circuits, and expand the regular model checking framework to account for the complexities of quantum systems.

Parameterized Verification One can carry out parameterized verification of quantum circuits by employing *cut-off* techniques. Empirical evidence suggests that concurrent programs often exhibit a *small model property*, indicating that analyzing only a small number of threads can suffice to capture the reachability of bad configurations. In practice, if any problematic behavior exists, it typically manifests in relatively small instances of the system.

Building on this insight, we can develop methods that leverage the small model property by restricting the verification process to a limited set of fixed-size system instances. Inspired by the *view abstraction* approach of [Abdulla et al., 2016], one can design a procedure that involves a fixed-point iteration where we apply gate operations on fixed-size states that effectively sample the quantum state. If the results are inconclusive, we can refine the abstraction by increasing the size of the sampled states. We can repeat this process until we either uncover a

specification violation or establish an invariant confirming the correctness of the circuit.

Regular Model Checking A major current limitation in applying regular model checking to quantum circuits is the lack of suitable frameworks to represent the semantics of quantum gates using transducers. One promising approach is to introduce a new class of weighted transducers, where each transition is labeled not only with input and output symbols—as in classical models—but also with a unitary matrix that captures the behavior of a quantum gate.

In existing quantum computing frameworks, circuit semantics are defined via matrix operations such as composition (matrix multiplication) and the Kronecker (tensor) product. These operations can be naturally expressed through the composition and product of transducers, leveraging their inherent compositional semantics. A central technical challenge in this context is computing the transitive closure of transducer relations. In general, such closures are either not computable or involve prohibitive computational complexity. To mitigate this, we can enhance the framework with acceleration and abstraction techniques, drawing on methods developed in prior work [Abdulla et al., 2007], in order to approximate or accelerate the computation of transitive closures.

Because both the (potentially infinite) sets of initial and target states can be encoded as automata, the regular model checking framework will provide a powerful and expressive foundation for verifying both parameterized and fixed-parameter quantum circuits.

6.3 Applications

We can apply the frameworks defined in Section 6.1 and Section 6.2 to solve verification problems of particular interest in quantum computing. We mention two problems, circuit equivalence, and amplitude measurement, as examples, which we consider in two different tasks.

Circuit Equivalence We need to develop a framework for checking circuit equivalence: given circuits C_1 and C_2, determine whether they define the same quantum function. The equivalence problem is relevant since we can use C_1 as a reference specification for a more complicated circuit C_2.

One can approach the solution to the equivalence problem based on the following observations. First, circuit equivalence can be verified by checking whether $C_1 C_2^\dagger$ implements the identity transformation, where $C_2^\dagger$ is obtained from C_2 by reversing the circuit (i.e., swapping inputs and outputs) and replacing each gate with its inverse. Since all quantum gates correspond to unitary matrices, their conjugate transposes give their inverses.

Second, to determine whether an n-qubit circuit is equivalent to the identity, we can test it on 2^n linearly independent *vectors*, verifying that each output vector matches its corresponding input. This criterion holds because the identity

matrix is the only transformation that maps all vectors to themselves. This step is challenging and will require the techniques such as the ones proposed above.

Third, due to the linearity of quantum operations, it is sufficient to test a maximal set of linearly independent vectors rather than exhaustively considering the entire state space.

Amplitude Verification In many quantum circuits, it is often essential to consider *relational specifications* that impose numerical constraints on the amplitudes found in the leaves of the trees representing quantum states. Such specifications allow us to express properties such as "Grover's circuit has >90% probability of finding a correct answer" or "the amplitude of the basis state is increased after executing the circuit C."

To this end, we can introduce *symbolic* quantum states, in which each computational basis state is associated with a numerical variable representing its amplitude. Relational specifications can then be naturally formulated as arithmetic constraints over these variables.

Verification of such specifications is achieved through symbolic execution of the quantum circuit, following the principles of symbolic execution described in [King, 1976]. We can then check whether all trees accepted by the resulting tree automaton (TA) satisfy the given property. Potentially, one can use a modified antichain-based algorithm for testing TA language inclusion [Abdulla et al., 2010]. As described above, these algorithms reduce the state space by pruning based on simulation relations over automaton states. In our setting, we can extend this pruning strategy to account for the numerical constraints associated with symbolic amplitudes, ensuring that the verification process respects quantum states' structural and quantitative aspects.

7 Conclusions

In addition to its early practical promise, the approach opens new research directions in automata theory, where quantum structures introduce novel mathematical challenges and opportunities. It establishes a connection between quantum program verification and automata, promoting new possibilities to exploit the richness of automata theory and automata-based verification in quantum computing. It is worth noting that the methodology is not just about catching mistakes but also about building trust in quantum computing systems as we move toward an era in which they might solve problems classical systems cannot.

In conclusion, the paper represents a confluence of disciplines, opening pathways for collaboration between automata theorists, quantum physicists, and software engineers. Looking ahead, this line of research opens up exciting opportunities. Could similar automata-based techniques be adapted to other aspects of quantum software engineering? Could this approach handle quantum programming languages' more abstract and flexible constructs?

References

[Abdulla, 2025] Abdulla, P. A. (2025). A symbolic approach to verifying quantum systems. To appear.

[Abdulla et al., 2025a] Abdulla, P. A., Chen, Y., Chen, Y., Chung, K., Holík, L., Lengál, O., Lin, J., Lo, F., Tsai, W., and Yen, D. (2025a). An automata-based framework for quantum circuit verification.

[Abdulla et al., 2025b] Abdulla, P. A., Chen, Y., Chen, Y., Holík, L., Lengál, O., Lin, J., Lo, F., and Tsai, W. (2025b). Verifying quantum circuits with level-synchronized tree automata. *Proc. ACM Program. Lang.*, 9(POPL):923–953.

[Abdulla et al., 2010] Abdulla, P. A., Chen, Y., Holík, L., Mayr, R., and Vojnar, T. (2010). When simulation meets antichains. In Esparza, J. and Majumdar, R., editors, *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*, volume 6015 of *Lecture Notes in Computer Science*, pages 158–174. Springer.

[Abdulla et al., 2007] Abdulla, P. A., Delzanno, G., Henda, N. B., and Rezine, A. (2007). Regular model checking without transducers (on efficient verification of parameterized systems). In Grumberg, O. and Huth, M., editors, *TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 721–736. Springer.

[Abdulla et al., 2016] Abdulla, P. A., Haziza, F., and Holík, L. (2016). Parameterized verification through view abstraction. *STTT*, 18(5):495–516.

[Abdulla et al., 2018] Abdulla, P. A., Sistla, A. P., and Talupur, M. (2018). Model checking parameterized systems. In Clarke, E. M., Henzinger, T. A., Veith, H., and Bloem, R., editors, *Handbook of Model Checking.*, pages 685–725. Springer.

[Bauer-Marquart et al., 2023] Bauer-Marquart, F., Leue, S., and Schilling, C. (2023). symqv: Automated symbolic verification of quantum programs. In Chechik, M., Katoen, J., and Leucker, M., editors, *Formal Methods - 25th International Symposium, FM 2023, Lübeck, Germany, March 6-10, 2023, Proceedings*, volume 14000 of *Lecture Notes in Computer Science*, pages 181–198. Springer.

[Chen et al., 2023a] Chen, Y., Chung, K., Lengál, O., Lin, J., Tsai, W., and Yen, D. (2023a). An automata-based framework for verification and bug hunting in quantum circuits. *Proc. ACM Program. Lang.*, 7(PLDI):1218–1243.

[Chen et al., 2023b] Chen, Y., Rümmer, P., and Tsai, W. (2023b). A theory of cartesian arrays (with applications in quantum circuit verification). In Pientka, B. and Tinelli, C., editors, *Automated Deduction - CADE 29 - 29th International Conference on Automated Deduction, Rome, Italy, July 1-4, 2023, Proceedings*, volume 14132 of *Lecture Notes in Computer Science*, pages 170–189. Springer.

[D'Hondt and Panangaden, 2006] D'Hondt, E. and Panangaden, P. (2006). Quantum weakest preconditions. *Mathematical Structures in Computer Science*, 16(3):429–451.

[King, 1976] King, J. C. (1976). Symbolic execution and program testing. 19(7).

[Perdrix, 2008] Perdrix, S. (2008). Quantum entanglement analysis based on abstract interpretation. In *International Static Analysis Symposium*, pages 270–282. Springer.

[Wulf et al., 2006] Wulf, M. D., Doyen, L., Henzinger, T. A., and Raskin, J.-F. (2006). Antichains: A New Algorithm for Checking Universality of Finite Automata. In *CAV'06*, volume 4144 of *LNCS*, pages 17–30. Springer.

[Yanofsky and Mannucci, 2008] Yanofsky, N. S. and Mannucci, M. A. (2008). *Quantum Computing for Computer Scientists*. Cambridge University Press, USA, 1 edition.

[Ying, 2012] Ying, M. (2012). Floyd-Hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 33(6):1–49.

[Yu and Palsberg, 2021] Yu, N. and Palsberg, J. (2021). Quantum abstract interpretation. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 542–558.