# On the Verification of Quantum Circuits

No Author Given

No Institute Given

**Abstract.** Quantum technology is progressing at a remarkable pace and holds the potential to transform numerous sectors both nationally and globally. As quantum systems become more advanced and increasingly prevalent, ensuring their correctness is of paramount importance. This underscores the urgent need for rigorous tools capable of analyzing and verifying their behavior. Nevertheless, the development of such verification tools presents substantial challenges. Distinct quantum phenomena—most notably superposition and entanglement—give rise to program behaviors that diverge fundamentally from those observed in classical computing. These characteristics lead to inherently probabilistic models and result in exponentially large state spaces, even for relatively simple systems.

In this tutorial we will illustrate initial steps that aims to tackle these challenges by leveraging the knowledge acquire in our community's on the verification of traditional systems. The objective is to design and implement novel verification frameworks that adapt the proven strengths of classical verification—such as succinct property specification, precise fault detection, automation, and scalability—to the quantum domain.

## 1 Introduction

We may get substantial added value when connecting two complementary areas of computer science. This applies particularly when adapting mature techniques developed in one area to solve complex problems that arise in another (tyupically) new area. The current paper illustrates one such a case. It describes the application of techniques developed in logic, automata, and symbolic verification to analyze the correctness of quantum circuits.

The current quest of quantum computing is achieving so called *quantum supremacy*, meaning that we reach a stage in the technology development where we can solve problems that are practically unsolvable using conventional computing. To that end, substantial effoprt is ongoing to implement quantum hardware and develop programming languages for quantum computers. In many applications, system correctness is of critical importance. For instance, identifying a subtle bug in a quantum circuit used for quantum chemistry simulations could prevent incorrect predictions about molecular interactions, which are critical for drug discovery and material design.

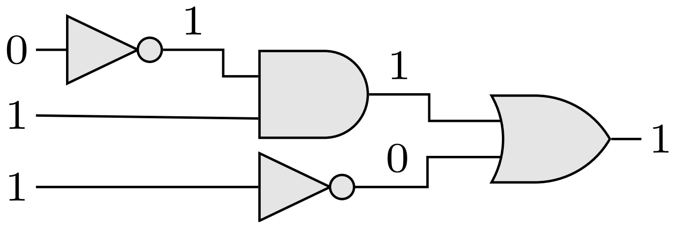
Given the complexity of quantum systems and the above mentioned correctness requirements, tools for analyzing and verifying quantum programs' correctness are of critical importance. However, building verification tools for quantum programs is a formiddable challenge. First, due their behaviors are probabilistic in nature, and their computational space is exponential in size. Furthermore, there is a fundamental difference in the manner in which conventional and quantum computers store information: conventional computers use digital bits (0s and 1s), while quantum computers use quantum bits (qubits) that probabilistically have values in nthe intgerval between 0 and 1.

The current paper reports on research in whose goal is adating the research community's vast experience in conventional program verification to develop tools for verifying quantum systems. Verification tools, in general, and for quantum systems in particular, would ideally have the following properties: (1) Flexibility: allows flexible specification of properties of interest, (2) Diagnostics: provides precise bug diagnostics, (3) Automation: Operates automatically, and (4) Scalability: scales efficiently to verify useful programs. Symbolic verification is one of the most successful techniques that satisfy the above criteria for conventional programs. Notable instances, are invariant certification (e.g., in the form of Hoare triples), and model checking. Despite the overwheling success achieved in the verification of conventional hardware and software, we are lacking an extension to the quantum realm. The principal reasonm isthe latter's unique mathematical structure and different operational principles.

In this work we describe an innovative application of automata theory to quantum circuit verification. More precisley, we combine we use tree automata-based symbolic representations for represeting quantum states. The class of tree we consider is tailored for quantum circuits, considerably extending the sacalbility of quantum circuit verification comnpared to existing techniques. As expected, when dealing with an entirely new application area, we we go back to the very basics of program verification. We consider classical Hoare triples $\{P\}C\{Q\}$, where $P$, the *pre-condition*, represents a set of initial quantum states, $Q$, the *post-conditions* represents a set of quantum *target* states, and $C$ is a quantum circuit. It uses symbolic verification to check the validity of the triple, ensuring that all executions of $C$ from states in $P$ result in states within $Q$. More precisely, we will resprent $P$ and $Q$ by a special form of tree automata, and provide algorithms to check whether we reach $Q$ from $P$ if we execute $C$. The framework is based on the observation that we can lift core concepts of classical verification, such as state-space exploration and symbolic reasoning, into the quantum setting.

Although this is the first attempt to use such techniques in the context of quantum computing, implementing the framework with a tool gives spectacular results. For instance, the tool manages to verify large circuits with up to 40 qubits and 141,527 gates or catch bugs injected into circuits with up to 320 qubits and 1,758 gates, while all exiting tools fail to handle such benchmarks.

This extension allows the verification of quantum circuits, offering a path to applying well-established classical paradigms in a domain where formal guarantees are critical.



## 2 Circuits

*Bits and Qubits* In classical computing, bits are the fundamental units of information. They can exist in only one of two possible states: 0 or 1. They may be conceptualized as switches that are either turned on or off. In contrast, quantum computing employs quantum bits, known as *qubits*, to represent and process information. Qubits can exist in a *superposition*, meaning they can be in state 0, state 1, or a combination of both simultaneously. Furthermore, a qubit can undergo a *measurement* process. A qubit's state can be described by two complex numbers, called *amplitudes*, $c_0$ and $c_1$, where $|c_0|^2$ represents the probability of measuring the state as 0, and $|c_1|^2$ the probability of measuring it as 1. We require that $|c_0|^2 + |c_1|^2 = 1$.

There are two common representations used to describe a qubit state. One is the vector notation, where the state is written as a column vector of length 2 (see **??**) with complex entries corresponding to the amplitudes of the respective basis states. To simplify notation, we often use the transpose of the column vector. This results in a row vector, which is more compact and easier to write. As a special case, we may refer directly to standard basis column vectors. The second representation is the Dirac notation, where the state is expressed as $c_0|0\rangle + c_1|1\rangle$. The classical states are special cases where either $c_0 = 1$ and $c_1 = 0$ or $c_0 = 0$ and $c_1 = 1$. In the contrext of quantum systems, we refer to theses states as the basis states $|0\rangle$ and $|1\rangle$.

We genberalize the above concepts to the case where we have multiple qubits. For $n$ bits we have $2^n$ states. For instance, when $n = 2$, we have four basis states $00, 01, 10, 11$, and a quantum state is of the form $c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|0\rangle + c_{11}|11\rangle$

The concepts Superposition allows quantum circuits to encode information more efficiently than classical circuits. Furthermore, qubits can be *entangled*: the state of one qubit is connected to another. Entaglement means the state of one qubit can't be described independently of the others. The state of one qbit may be possible to derive from the state of another qubit, even if when physicallyu far away.

We will represent qu(bits) by (column) matrices

$$\begin{bmatrix}0\\1\end{bmatrix} \quad \begin{bmatrix}1\\0\end{bmatrix} \quad \begin{bmatrix}\sqrt{\frac{1}{2}}\\ \\ \sqrt{\frac{1}{2}}\end{bmatrix} \quad \begin{bmatrix}\frac{1}{3}\\ \\ \frac{\sqrt{8}}{3}\end{bmatrix} \quad \begin{bmatrix}c_0\\ \\ c_1\end{bmatrix} [c_0 \quad c_1]^{\mathtt{T}}$$

More egenrally, we will work with multiple (qu)bits. In the case where we have two qubits the systems wiill have four states. The basis states are given by

$$[1\,0\,0\,0]^{\mathtt{T}} \quad [0\;\;1\,0\,0]^{\mathtt{T}} \quad [0\,0\,1\,0]^{\mathtt{T}} \quad [0\,0\,0\,1]^{\mathtt{T}}$$

A quantum state will have the form

$$[c_0 \;\; c_1 \;\; c_2 \;\; c_3]^{\mathtt{T}} \quad \begin{bmatrix}\frac{1}{4} & \frac{3}{4} & \frac{1}{\sqrt{8}} & \frac{1}{2}\end{bmatrix}^{\mathtt{T}}$$

*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. Quantum gates like the Hadamard and CNOT are the building blocks of quantum circuits, similar to classical logic gates. They manipulate qubits through unitary operations, allowing for quantum algorithms. Each gate performs a specific transformation, enabling superposition, entanglement, and measurement, thus enabling more complex computations compared to classical gates.

*States* The state of a sequence of bits is often represented by listing their values. For example, the input state in the figure is 011, indicating the values of the three input bits. Generally, $n$ bits can have $2^n$ possible states.

In quantum computing, bits and states have a richer structure. A *basis state* for a quantum bit is one of the classical states 0 or 1. A qubit can be in a basis state or, more generally, in a *superposition* of the basis states, represented mathematically as: $[|\psi> = \alpha|0> + \beta|1>]$. Here, $\alpha$ and $\beta$ are complex coefficients that determine the probability of measuring the qubit in either state. The probabilities are given by: Probability of measuring $|0>$: $(|\alpha|^2)$ Probability of measuring $|1>$: $(|\beta|^2)$ The condition $(|\alpha|^2 + |\beta|^2 = 1)$ must hold to ensure valid probabilities. In a system with $n$ qubits, there are $2^n$ possible basis states. In general, a quantum state is a superposition of a set of basis state, representing a probability distribution over the possible $2^n$ bit-assignments. A quantum state $q$ assignes to each basis state $b$ a complex number $\alpha_b$ called the *amplitude* of $b$ in $q$. Intuitively, if we measure $q$ then the probability that we observe $b$ is given by the square $\alpha_b^2$ iof thbe amplitude. The a The left side of Fig. 1 shows a quantum state with three qubits. Each row of the table gives the amplitude of one basis state.

*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. Describe quantum states

The basis states represent the simplest forms of qubit states: —0⟩: Represents the state where the qubit is definitively in the "0" position. —1⟩: Represents the state where the qubit is definitively in the "1" position. These states are often referred to as computational basis states. 3. Superposition

## 3 Trees

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

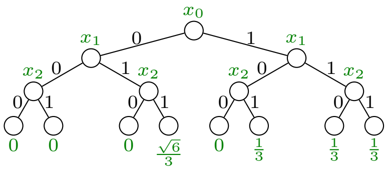


Fig. 1: A quantum state with three qubits and its representation

A key concept in our framework is a symbolic representation of quantum states and sets of quantum states. We use *tree automata* to instantiate the symbolic encoding. Specifically, we encode a quantum state, with $n$ qubits as a perfect binary tree with depth $n$. Each level of the tree corresponds to a single qubit. We might refer to the level corresponding to qubit $x$ as the $x$-level, and the nodes at the $x$-level as the $x$-nodes. For an $x$-node in the tree, the left and right sub-trees corresponds to $x$ being in the values 0 and 1 respectively. Each path in the tree, from the root to a leaf, represents a computational basis state. The leaves represent the complex probability amplitudes of these basis states.

Consider Fig. 1. The left part shows a quantum state comprising nthree qubits. On the right side, we show its tree representation.

How about BDDs and classical tree automata?

In 2, we depict the result of applying the CNOT gate on the quantum state represented by the tree $T_1$. We obtain a new state represented, by the tree $T_2$, where we swap the left- and righ-subtrres of each node at level 2 of the tree.

## 4 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

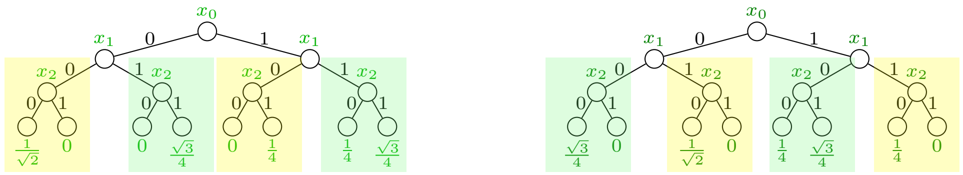


Fig. 2: Applying the NOT gate to the second qubit.

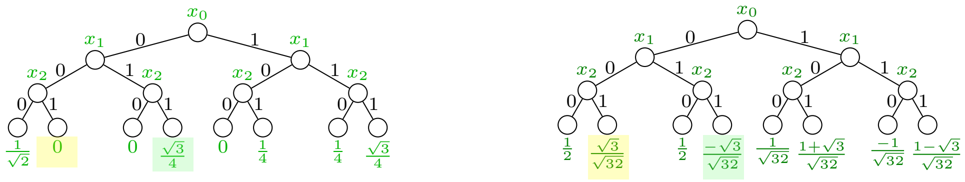


Fig. 3: Applying the H gate to the second qubit.

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?