Verifying Repeat-until-Success Protocols using Automata

JYUN-AO LIN✉, National Taipei University of Technology, Taiwan
YU-FANG CHEN, Academia Sinica, Taiwan
JAKUB HAVLÍK, Brno University of Technology, Czech Republic
ONDŘEJ LENGÁL, Brno University of Technology, Czech Republic
FANG-YI LO, Academia Sinica, Taiwan
WEI-LUN TSAI, National Taiwan University, Taiwan
YOU-JIE WU, National Taipei University of Technology, Taiwan

Repeat-until-success (RUS) protocols implement single-qubit unitaries using measurement, classical control, and unbounded looping. Verifying their functional correctness is challenging due to the combination of probabilistic branching, unbounded looping, and the need to reason about all input states. In this paper, we develop a fully automated framework for verifying the functional correctness of these protocols. The framework is based on viewing quantum states as *trees* and sets of quantum states as sets of trees, which can be represented using tree automata. The particular automata model that we use are *level-synchronized tree automata* (LSTAs), in which nondeterminism is labelled by a *choice*. Since we can map a sequence of choices to a particular tree (and therefore a quantum state) in the language of an LSTA, we can use the choice-sequence semantics to track input-output correspondence (which input quantum state got transformed into which output quantum state) and enable relational verification. To deal with reasoning about infinitely many quantum states, we prove a three-test theorem, which reduces verifying correctness of RUS protocols to testing correctness on finitely many inputs, enabling automatic invariant synthesis and decidable verification. We implemented our approach and identified previously unreported bugs in the RUS literature.

CCS Concepts: • **Hardware → Quantum computation**; • **Theory of computation → Tree languages**; • **Software and its engineering → Formal software verification**.

Additional Key Words and Phrases: quantum circuits, tree automata, verification

ACM Reference Format:
Jyun-Ao Lin, Yu-Fang Chen, Jakub Havlík, Ondřej Lengál, Fang-Yi Lo, Wei-Lun Tsai, and You-Jie Wu. 2026. Verifying Repeat-until-Success Protocols using Automata. *Proc. ACM Program. Lang.* 10, OOPSLA2, Article 374 (October 2026), 26 pages. https://doi.org/10.1145/3839506

1 Introduction

Quantum computing has moved beyond the stage where quantum circuits are viewed purely as fixed unitary transformations. Modern quantum platforms increasingly support *dynamic circuits*, that is,

Authors' Contact Information: Jyun-Ao Lin, National Taipei University of Technology, Innovation Frontier Institute of Research for Science and Technology and Department of Computer Science and Information Engineering, Taipei, Taiwan, jalin@ntut.edu.tw; Yu-Fang Chen, Academia Sinica, Institute of Information Science, Taipei, Taiwan, yfc@iis.sinica.edu.tw; Jakub Havlík, Brno University of Technology, Faculty of Information Technology, Brno, Czech Republic, xhavlij00@stud.fit.vutbr.cz; Ondřej Lengál, Brno University of Technology, Faculty of Information Technology, Brno, Czech Republic, lengal@fit.vutbr.cz; Fang-Yi Lo, Academia Sinica, Institute of Information Science, Taipei, Taiwan, lofangyi@gmail.com; Wei-Lun Tsai, National Taiwan University, Graduate Institute of Electronics Engineering, Taipei, Taiwan, alan23273850@gmail.com; You-Jie Wu, National Taipei University of Technology, Innovation Frontier Institute of Research for Science and Technology, Taipei, Taiwan, t113C53056@ntut.edu.tw.

ACM 2475-1421/2026/10-ART374
https://doi.org/10.1145/3839506

computations that combine quantum evolution with mid-circuit measurement, feedforward unitary transformations, reset, and conditional control [13, 15, 20]. These capabilities are central to both near-term and fault-tolerant quantum computing, because they allow quantum programs to react to measurement outcomes during execution rather than only at the end. As a result, the verification target is no longer just a static circuit, but a quantum *program* that may contain branching, looping, and interaction between classical and quantum states.

A particularly important instance of this adaptive programming style is the *Repeat-until-Success* (RUS) paradigm [7, 19]. In an RUS protocol, one executes a circuit, measures an ancilla qubit, and either terminates on a success outcome or performs a recovery step and repeats the process. This pattern allows certain operations, especially non-Clifford gates and small-angle rotations, to be implemented probabilistically with lower resource cost than straightforward deterministic decompositions. For this reason, RUS constructions have been studied extensively for fault-tolerant single-qubit gate synthesis, where they can reduce the expected implementation cost of non-Clifford operations, especially the expected T-count. Beyond the use for hardware optimization, the same measurement-and-feedback principle has also been explored in quantum machine learning (QML) [17], where mid-circuit measurement and conditional repetition introduce nonlinear activation-like behavior, increasing the expressive capacity of variational quantum models, or in the context of weakly measured loops [6].

Despite their usefulness, RUS protocols are difficult to verify. Their behavior depends on both quantum amplitudes and classical measurement outcomes, and correctness must account for all possible loop iterations. This creates a verification problem that is qualitatively different from checking the semantics of a unitary circuit. One must show not merely that some branch produces the intended result, but that *every terminating execution* returns the correct output state for *every* input state, while every failing branch restores the program to a configuration from which the protocol may safely continue.

From a programming-languages perspective, this difficulty is part of a broader challenge in reasoning about quantum programs with classical control. RUS protocols are a canonical example because they combine all of the ingredients that make verification hard: measurement, probabilistic branching, unbounded repetition, and a functional correctness requirement relating each input state to its corresponding output state. Extensional set-based specifications used in circuit-level reasoning can be too coarse for this setting, since they do not by themselves track the correspondence between individual inputs and outputs across measurement branches and loop iterations. For example, both $C = \mathrm{Id}$ and $C = \mathrm{X}$ satisfy the extensional specification $\{|0\rangle, |1\rangle\}\ C\ \{|0\rangle, |1\rangle\}$, even though Id preserves each basis state while X swaps them. The specification therefore fixes the output set but not the correspondence between individual inputs and outputs.

To address this problem, we develop an automated verification technique for RUS protocols on top of the automata-based framework of [3, 9–12]. Our work is motivated by an unexpected finding when analyzing published RUS constructions. In the example shown in Fig. 1, a protocol from [19] is supposed to implement the unitary $\frac{3I+2iZ}{\sqrt{13}}$. However, our analysis shows that the unitary that it actually implements is $\frac{3I-2iZ}{\sqrt{13}}$. This is not a minor error; it results in a quite different functionality. Since the example comes from a well-known and widely used paper, such an error could easily be adopted into compilers or synthesis tools without being noticed. This could, in turn, lead to hard-to-detect incorrect results in quantum programs. This concrete example of a bug justifies the need for reliable automated verification of RUS protocols.

Our verification technique stands on two keystones. The first one, called the *three-test theorem* (Theorem 6.1), shows that, for a two-qubit RUS protocol, it is sufficient to check their behavior on

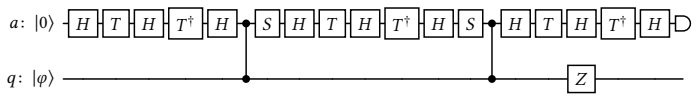


Fig. 1. An RUS circuit from [19, Figure 10(a)] intended to implement the unitary $\frac{3I+2iZ}{\sqrt{13}}$ on the data qubit q (when the ancilla a is measured to be 0). The real unitary implemented by the circuit is, however, $\frac{3I-2iZ}{\sqrt{13}}$

the inputs $|00\rangle$, $|01\rangle$, and $|0+\rangle$[1] in order to verify their correctness. This gives a compact functional specification and explains why weaker two-state checks are insufficient. The other keystone is lifting this characterization into an automata-based verification framework built on top of *level-synchronized tree automata* (LSTAs) [3, 11]. The framework is based on viewing quantum states as *trees* and sets of quantum states as sets of trees, which can be represented using tree automata. LSTAs are a particular tree automata model in which nondeterminism is labelled by a *choice*. We use the fact that a sequence of choices can be mapped to a particular tree (and therefore a quantum state) in the language of an LSTA, and we can use the *choice-sequence semantics* to track input-output correspondence (which input quantum state got transformed into which output quantum state). We introduce *choice-aware language inclusion*, which is a refinement of the language inclusion operation that takes into account the choice-sequence semantics, thus enabling *relational verification*, which is essential for verifying functional correctness of RUS protocols.

These two key ideas yield a fully automated method for verifying RUS protocols. The loop invariant required for the while-loop proof rule can be synthesized automatically from the precondition automaton. The remaining verification conditions reduce to automata-theoretic inclusion checks, including an up-to-scale variant needed to account for the non-normalized states produced by measurement. Using this method, we are able to validate corrected RUS constructions automatically and reject incorrect ones that evaded previous checks (see [11]).

Contributions. The contributions of this paper are as follows.

- We prove a *three-test theorem* (Theorem 6.1) for two-qubit RUS protocols and discuss how the same linearity argument naturally extends to $(2^n + 1)$-state verification of n-qubit targets and multiple ancilla branches. The theorem shows that a candidate circuit has the correct RUS form if it behaves appropriately on the three inputs $|00\rangle$, $|01\rangle$, and $|0+\rangle$ (we also give a justification why weaker criteria are insufficient). This yields a compact functional specification for RUS verification.
- We introduce *choice-sequence semantics* for LSTAs (Section 3.3). Choice sequences record the synchronized nondeterministic decisions made during accepting runs. We prove that the LSTA transformations corresponding to quantum gates and measurements preserve these sequences (Theorem 4.1), thereby maintaining a structured correspondence between input and output states.
- We define *choice-aware language inclusion* (Section 4.2), a new automata-theoretic relation for comparing LSTAs while respecting choice sequences. We also develop an up-to-scale variant that accommodates the global scaling factors arising from measurement.
- We apply the framework to the *automated verification* of Repeat-until-Success protocols. The loop invariant can be synthesized automatically from the precondition automaton, yielding a fully automated verification procedure that detects incorrect RUS constructions while successfully verifying fixed ones.

[1] $|+\rangle$ denotes the quantum state $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$.

2 Overview

This section gives a high-level overview of the verification technique developed in this paper. We illustrate the key ideas of our framework using a small running example based on an RUS protocol, with details given later in the paper.

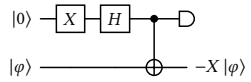
Our goal is to verify the *functional correctness* of such protocols: for every input state $|\varphi\rangle$, if the program terminates, it should produce the state $U|\varphi\rangle$ for a given target unitary U. The central difficulty is that RUS protocols contain measurements and loops, which introduce both probabilistic branching and potentially unbounded execution.

The approach taken in this work proceeds in the following steps.

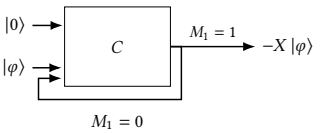
(1) (Pure) quantum states are represented as perfect binary trees whose leaves store complex amplitudes. Sets of quantum states are represented symbolically using Level-Synchronized Tree Automata (LSTAs).
(2) Quantum gates and measurements are interpreted as transformations of these LSTAs.
(3) Correctness is verified using a new relation called *choice-aware language inclusion*, which refines standard language inclusion by tracking the correspondence between input and output trees.

2.1 Running Example of an RUS Protocol

We illustrate the main ideas of the paper using a small RUS protocol. The purpose of the protocol is to implement the single-qubit unitary $U = -X = \left[\begin{smallmatrix} 0 & -1 \\ -1 & 0 \end{smallmatrix}\right]$ on a data qubit. The protocol on the right-hand side of Fig. 2 operates on two qubits. The first qubit is an ancilla, which is initialized to $|0\rangle$ at the beginning of each iteration. The second qubit is the data qubit, which contains the input state $|\varphi\rangle$. One iteration of the protocol applies the circuit C (including the measurement) shown in the left-hand side of Fig. 2. For later reference, we write F for the linear operator (loop body) implemented by this circuit before the measurement is performed.



(a) The RUS circuit C where the output $-X|\varphi\rangle$ is conditioned on the measurement outcome 1



(b) The high-level structure of an RUS protocol

Fig. 2. An RUS circuit C and the corresponding protocol

When the measurement outcome is 1, the iteration is successful, and the data qubit is transformed to $-X|\varphi\rangle$. Conversely, when the outcome is 0, the iteration fails, and the protocol repeats. In this specific illustrative example, the failure measurement directly leaves the data qubit in its original state $|\varphi\rangle$, allowing the next iteration to begin immediately from the initial quantum state $|0\rangle|\varphi\rangle$. We note that this seamless restoration is not generally guaranteed; the formal structure of a general RUS protocol may also contain execution of a non-trivial recovery circuit after a failed iteration, which is detailed in Section 3.2.

Thus, the circuit together with the measurement forms a probabilistic procedure for implementing the target unitary U. At the program level, this behaviour is naturally viewed as a while-loop:

$$\mathrm{X}_1;\mathrm{H}_1;\mathrm{CNOT}_2^1;\texttt{measure ancilla};\ \textbf{while}\ M_1 = 0\ \textbf{do}\ \{\mathrm{X}_1;\mathrm{H}_1;\mathrm{CNOT}_2^1;\texttt{measure ancilla}\}, \quad (1)$$

here, $M_1 \in \{0, 1\}$ denotes the classical outcome of the immediately preceding ancilla measurement. This example already exhibits two features that make the verification problem interesting. First, the

program contains a measurement, so its control flow depends on probabilistic quantum behaviour. Second, the program contains an unbounded loop, since the number of attempts is not known in advance.

To verify the functional correctness of the protocol, it is not enough to show that some successful executions produce the desired output. Rather, we must show that whenever the protocol terminates, the resulting state of the data qubit is exactly $-X|\varphi\rangle$ for the original input $|\varphi\rangle$.

From a program-verification perspective, this requirement can be viewed as a Hoare-style specification: starting from the state $|0\rangle|\varphi\rangle$, every terminating execution should produce the state $|1\rangle(-X|\varphi\rangle)$. We formalize this requirement in Section 5.1 as a *functional specification triple*

$$\{|0\rangle|\varphi\rangle\} \ \text{RUS} \ \{|1\rangle(-X|\varphi\rangle)\}, \tag{2}$$

which associates each input state with the corresponding output state produced upon termination.

In the remainder of this overview we explain how this property can be established automatically using level-synchronized tree automata.

2.2 Representing Quantum States Using Trees

First, let us describe how our framework represents (pure) quantum states using a tree structure. An n-qubit state $|\varphi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x |x\rangle$ can be viewed as a vector with 2^n complex amplitudes. Instead of storing this vector explicitly, we organize the amplitudes in a perfect binary tree of height n.[2] Each internal node is labelled by the name of a qubit while leaf nodes store the complex amplitudes. The left and right children of a node represent the cases where the corresponding qubit is 0 or 1. For instance, Fig. 3 illustrates the tree representations of the quantum states $|00\rangle, |01\rangle$, and $|0+\rangle$ respectively.

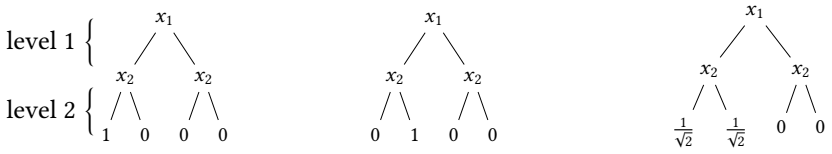


Fig. 3. Tree representations of the quantum states $|00\rangle, |01\rangle$, and $|0+\rangle$

While a single tree represents one quantum state, verification requires reasoning about sets of states. In our framework, such sets are represented by LSTAs. Fig. 4 shows the LSTA $\mathcal{A}$ used to represent the three states $|00\rangle, |01\rangle$ and $|0+\rangle$. Each accepting run generates a perfect binary tree of height 2 whose leaves store the amplitudes of a two-qubit state. In this representation, the first level of the tree corresponds to the ancilla qubit and the second level corresponds to the data qubit. The root state is r, and the first two transitions describe the first level of the tree. Two transitions are available at the first level (the first column in Fig. 4). Both transitions produce the subtree rooted at state q_0 on the right branch, which fixes the first qubit to the value 0. The run through state q_1 generates the basis states $|0\rangle, |1\rangle$ of the second qubit. The alternative run through state q_2 generates a superposition state $|+\rangle$.

An essential feature of LSTAs is the *level-synchronization* condition. Each transition carries a set of *choices*. During an accepting run, the transitions used at the same level of the tree must have compatible choice sets ; that is, their sets must share at least one common element. This mechanism can be observed in the transition $q_0 \xrightarrow{\{2,3,4\}} x_2(q_5, q_5)$. At level 2, because the choice set $\{2, 3, 4\}$

[2]A perfect tree is a tree where all branches have the same length.

$$\underbrace{\begin{array}{l} r \xrightarrow{\{1\}} x_1(q_1, q_0), \\ r \xrightarrow{\{2\}} x_1(q_2, q_0), \end{array}}_{\text{level 1}} \quad \underbrace{\begin{array}{l} q_1 \xrightarrow{\{2\}} x_2(q_3, q_5) \\ q_1 \xrightarrow{\{3\}} x_2(q_5, q_3) \\ q_2 \xrightarrow{\{4\}} x_2(q_4, q_4) \\ q_0 \xrightarrow{\{2,3,4\}} x_2(q_5, q_5) \end{array}}_{\text{level 2}} \quad \underbrace{\begin{array}{l} q_5 \xrightarrow{\{1\}} 0 \\ q_3 \xrightarrow{\{1\}} 1 \\ q_4 \xrightarrow{\{1\}} \frac{1}{\sqrt{2}} \end{array}}_{\text{leaf amplitudes}} \tag{3}$$

Fig. 4. An LSTA $\mathcal{A}$ recognizing three quantum states $|00\rangle$, $|01\rangle$, and $|0+\rangle$

contains both 2 and 3, the transition from q_0 is compatible with either transition from q_1. Hence the synchronization condition can be satisfied with choice value 2 or 3 at this level.

A small modification illustrates the role of this choice set. Suppose that we replace the transition $q_0 \xrightarrow{\{2,3,4\}} x_2(q_5, q_5)$ rooted at q_0 by the transition $q_0 \xrightarrow{\{2,4\}} x_2(q_5, q_5)$. Since $\{2, 4\}$ does not intersect with $\{3\}$, the run corresponding to the transition $q_1 \xrightarrow{\{3\}} x_2(q_5, q_3)$ becomes forbidden due to the level-synchronization condition. The modified LSTA therefore recognizes only the two trees corresponding to $|00\rangle$ and $|0+\rangle$.

In general, several transitions may be used at the same level of the tree during a run of the automaton. Each of these transitions carries a set of choices. The *level-synchronization condition* requires that these sets are compatible, meaning that they share at least one common element. The choices that remain possible at that level are precisely those contained in the intersection of the sets. E.g., the sequence $(\{1\}, \{2\}, \{1\})$ corresponds to the quantum state $|00\rangle$, the sequence $(\{1\}, \{3\}, \{1\})$ corresponds to $|01\rangle$, and the sequence $(\{2\}, \{4\}, \{1\})$ corresponds to $|0+\rangle$.

Thus, at every level of the run we obtain a set of choices that are jointly compatible with all transitions used at that level. As the run proceeds down the tree, these sets form a sequence associated with the run. We call this sequence the *choice sequence* of the run.

Section 4.2 will show that the quantum operations used in the RUS protocol transform LSTAs in such a way that these choice sequences are preserved. As a result, each input state represented by an LSTA remains uniquely associated with a corresponding output state after executing the quantum program.

2.3 Executing Quantum Operations on LSTAs

To reason about the execution of a quantum program, we must understand how quantum operations transform the sets of states represented by LSTAs. In [4, 11], the authors introduced algorithmic constructions that transform an LSTA according to the application of quantum gates and measurement operations. These constructions allow the symbolic execution of quantum programs directly at the level of automata.

In the present paper we reuse these transformations without modification. What is new here is an additional structural property (Theorem 4.1): the transformations preserve the choice sequences of accepting runs. Intuitively, although the constructions may introduce new states and transitions, the synchronization mechanism of the automaton remains intact.

This observation will play a central role in the verification framework developed in the later sections. Because choice sequences are preserved during program execution, we can relate the states represented by the input automaton to the states represented by the output automaton in a precise and structured way.

2.4 Choice-Sequences and Verification of Functional Correctness

We return to the RUS protocol from Section 2.1 and explain how it is verified in our framework. Recall that the protocol is intended to realize the $-X$ gate on the data qubit. Since the ancilla is initialized to $|0\rangle$ and a successful execution terminates with measurement outcome 1, the desired behaviour can be expressed by the functional specification Eq. (2).

The protocol itself has the structure of a while-loop as Eq. (1). One iteration C applies the circuit F, measures the ancilla, and either terminates or repeats. Following the usual reasoning principles for while-loops, the correctness of the program can be established by verifying three specification triples:

- $\{\!|\mathcal{A}_{\text{pre}}|\!\}\ \ F\ \ \{\!|\mathcal{I}|\!\}$, the invariant holds before entering the loop,
- $\{\!|M_1^{=0}(\mathcal{I})|\!\}\ \ F\ \ \{\!|\mathcal{I}|\!\}$, the invariant is preserved by a failing iteration of the loop, and
- $\{\!|M_1^{=1}(\mathcal{I})|\!\}\ \ \texttt{Skip}\ \ \{\!|\mathcal{A}_{\text{post}}|\!\}$, when the loop terminates, the invariant implies the desired postcondition.

At first sight, this verification problem appears difficult. The specification requires correctness for *every* input state $|\varphi\rangle$ of the data qubit, and therefore involves infinitely many possible inputs. Checking the specification directly would require reasoning about the behaviour of the circuit on all such states. In principle, one could attempt to represent this infinite family of states symbolically. For example, an LSTA could use symbolic values on the leaves of the tree to represent an arbitrary state of the data qubit. However, reasoning about such symbolic states quickly becomes intractable. The validity checks required by our framework would generate first-order constraints involving a mixture of quantified nonlinear integer arithmetic and nonlinear real arithmetic. The satisfiability problem for such formulas is undecidable in general, and in practice, SMT solvers typically fail to handle them effectively.

Fortunately, a key observation allows us to reduce this infinite verification problem to a finite one. Informally, Theorem 6.1 shows that it suffices to check the behaviour of the protocol on three representative input states $|00\rangle, |01\rangle$ and $|0+\rangle$. Thus, the precondition of the specification can be represented by the LSTA $\mathcal{A}_{\text{pre}}$ (which is the one illustrated in Fig. 4) that recognizes exactly these three states. Similarly, the postcondition $\mathcal{A}_{\text{post}}$ consists of the three corresponding output states $-|11\rangle, -|10\rangle$ and $-|1+\rangle$. More precisely, $\mathcal{A}_{\text{post}}$ is constructed so that its accepting runs carry matching choice sequences, ensuring that each output state recognized by $\mathcal{A}_{\text{post}}$ is aligned with the corresponding input state in $\mathcal{A}_{\text{pre}}$. Using the automata-based semantics described in Section 4.1, the circuit transformation F and the measurement operations can be applied directly to the LSTA $\mathcal{A}_{\text{pre}}$. This yields automata that represent the sets of states appearing in the verification conditions for the while-loop.

In particular, the Theorem 6.1 ensures that the circuit F has the uniform form $F(|0\rangle\,|\varphi\rangle) = a\,|1\rangle\,U\,|\varphi\rangle + b\,|0\rangle\,R\,|\varphi\rangle$ for every input $|0\rangle\,|\varphi\rangle$. Consequently, after one application of F, the system reaches a state of the form. It follows that the states produced immediately after applying the circuit F form a natural loop invariant. In our framework, this invariant can be obtained automatically by applying the circuit transformation to the precondition automaton, namely $F(\mathcal{A}_{\text{pre}})$. Thus, the invariant does not need to be supplied by the user; it can be *synthesized automatically* from the precondition.

The correctness of the protocol then reduces to checking three specification triples. In our framework, each of these checks becomes a problem of comparing two LSTAs that represent the corresponding sets of quantum states.

However, because we are verifying *functional correctness*, it is not sufficient to compare the automata merely as sets of states. Instead, we must ensure that each input state is matched with the correct output state produced by the program. To capture this requirement, we introduce

choice-aware language inclusion in Section 5. Recall that every accepting run of an LSTA carries a choice sequence describing how the run is synchronized across levels. These choice sequences provide a natural way to track the correspondence between input and output states represented by the automata. Moreover, as mentioned in the previous subsection, the automata transformations that model quantum gate applications preserve these choice sequences, so the correspondence between runs is maintained throughout the execution of the program. Choice-aware inclusion, therefore, compares two automata while preserving this correspondence. In Section 5.2, we present decision procedures for computing such choice-aware inclusion checks.

Another technical issue arises from measurement. The automata transformations for measurement operations [11] do not normalize the amplitudes of quantum states produced by measurement. Instead, states are compared *up to a nonzero scalar factor*. Accordingly, we introduce in Section 5.3 an up-to-scale variant of choice-aware inclusion that identifies states differing only by such a factor.

Putting these ingredients together, the verification of the RUS protocol proceeds as follows. First, the three-test theorem Theorem 6.1 reduces the functional specification to a finite set of test states. These states are represented symbolically by the LSTA $\mathcal{A}_{\text{pre}}$ describing the precondition. Next, the circuit and measurement operations are applied to the automaton using the automata-based semantics, yielding automata that describe the reachable states of the program. The loop invariant is synthesized automatically as $F(\mathcal{A}_{\text{pre}})$. Finally, the correctness of the protocol reduces to checking choice-aware inclusion, together with its up-to-scale variant, between the LSTAs arising from the two measurement branches of the invariant and the automata representing the invariant and the postcondition.

The remainder of the paper develops this framework formally. Section 3 reviews the necessary background on quantum computation and repeat-until-success protocols, introduces the syntax and semantics of level-synchronized tree automata, and defines the notion of *choice sequences* associated with accepting runs of an automaton. Section 4 presents the LSTA transformations corresponding to quantum operations and proves that these transformations preserve choice sequences. Section 5 defines *choice-aware language inclusion* and its up-to-scale variant, and presents decision procedures for checking these relations. Section 6 establishes the three-test theorem and shows how it leads to an automated verification framework for RUS protocols. Finally, Section 7 reports the experimental evaluation of our approach and Section 8 concludes the paper.

3 Preliminary

3.1 Quantum States and Gates

For $n \in \mathbb{N}_0$, the state space of n qubits is the complex vector space $\mathbb{C}^{2^n}$. Its standard basis is indexed by bitstrings $v \in \{0,1\}^n$ and is called the *computational basis*. An n-qubit pure state is a unit vector $|\varphi\rangle \in \mathbb{C}^{2^n}$. We write $|\varphi\rangle\,(v)$ for the coordinate of $|\varphi\rangle$ corresponding to basis index v; thus $|\varphi\rangle$ assigns a complex amplitude to each bitstring $v \in \{0,1\}^n$.

A quantum gate acting on n qubits is represented by a unitary linear operator $U : \mathbb{C}^{2^n} \to \mathbb{C}^{2^n}$, that is, a matrix satisfying $U^\dagger U = I$, where $U^\dagger$ denotes the conjugate transpose of U and I is the identity matrix. Its action on a state $|\varphi\rangle$ is given by matrix-vector multiplication $|\varphi'\rangle = U\,|\varphi\rangle$.

A two-outcome *measurement* of the ith qubit is specified by linear operators $M_i^{=0}, M_i^{=1} \in \mathbb{C}^{2^n \times 2^n}$ satisfying the condition $(M_i^{=0})^\dagger M_i^{=0} + (M_i^{=1})^\dagger M_i^{=1} = I$. Given a state $|\varphi\rangle$, the outcome $b \in \{0,1\}$ occurs with probability $\| M_i^{=b}\,|\varphi\rangle \|^2$, and the corresponding post-measurement state is $|\varphi_b\rangle = \frac{M_i^{=b}|\varphi\rangle}{\| M_i^{=b}|\varphi\rangle \|}$.

3.2 Repeat-until-Success Protocols

Repeat-until-success (RUS) protocols are a technique for synthesizing unitaries using a small quantum circuit that is executed repeatedly until a desired measurement outcome is obtained. In this work, we consider the standard variant studied in the RUS literature [19], where a two-qubit circuit is used to synthesize a single-qubit unitary. One qubit serves as the data qubit, which carries the input state, and the other serves as an ancilla. At the beginning of each attempt, the ancilla is initialized to $|0\rangle$.

The protocol consists of two components: The first component is a two-qubit circuit F. In practice, such a circuit is composed of unitary gates drawn from a fixed finite universal gate set, such as the Clifford+T gate set commonly used in fault-tolerant quantum computing. The second component is a measurement of the ancilla qubit in the computational basis. We view the RUS circuit as the linear operator F implemented by the unitary portion of the protocol preceding the measurement. After applying F, the ancilla (qubit 1) is measured using the operators $M_1^{=b}$, $b \in \{0,1\}$. The measurement outcome determines whether the current attempt succeeds.

At the program level, an RUS protocol can be viewed as a loop that repeatedly applies the circuit until the measurement indicates success:

$$F; \texttt{measure ancilla}; \ \textbf{while}\ \texttt{outcome}\ b = 0\ \textbf{do}\ \{R^\dagger; F; \texttt{measure ancilla};\}.$$

Let $|\varphi\rangle \in \mathbb{C}^2$ be the initial state of the data qubit. Since the ancilla is initialized to $|0\rangle$, the initial joint state of the system is $|0\rangle\,|\varphi\rangle$. One execution of the circuit F produces a state of the form $F(|0\rangle\,|\varphi\rangle) = a\ |1\rangle\ U\,|\varphi\rangle + b\ |0\rangle\ R\,|\varphi\rangle$, for non-zero complex constants a, b (where $|a|^2 + |b|^2 = 1$) and single-qubit unitaries U and R. If the measurement outcome is 1, the state collapses to $|1\rangle\ U\,|\varphi\rangle$ and the protocol terminates successfully. If the outcome is 0, the state collapses to $|0\rangle\ R\,|\varphi\rangle$. Applying the repair operation $R^\dagger$ restores the data qubit to $|\varphi\rangle$, so that the next iteration again begins from the configuration $|0\rangle\,|\varphi\rangle$.

The protocol is said to *synthesize* the unitary U if every terminating execution produces the state $|1\rangle\ U\,|\varphi\rangle$ for the original input $|\varphi\rangle$, up to a global scalar factor.

3.3 Level-Synchronized Tree Automata

3.3.1 Tree. Let $\mathbb{N}$ be the set of positive integers and $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. For $n \in \mathbb{N}_0$, write

$$\{0,1\}^{\le n} \ := \bigcup_{0 \le i \le n} \{0,1\}^i,$$

and let ε denote the empty word. For words $v, w \in \{0,1\}^*$, we write $v.w$ for their concatenation. In particular, if $b \in \{0,1\}$, then $v.b$ denotes the word obtained by appending b to v.

A *perfect binary tree* of height n over a nonempty set Σ is a function

$$T\colon \{0,1\}^{\le n} \to \Sigma.$$

The elements of $\{0,1\}^{\le n}$ are the nodes of T. The root is ε. A node v has height $\mathrm{ht}(v) := |v|$, its word length. Nodes with $|v| < n$ are *internal* and have exactly two children, $v.0$ and $v.1$. Nodes with $|v| = n$ are leaves and have no children. We refer to n as the height of T.

In this work, perfect binary trees of height n serve as semantic objects representing pure n-qubit states. The leaves $\{0,1\}^n$ are identified with computational basis indices, and each leaf stores a complex amplitude. Thus a tree T induces canonically a vector $\varphi_T \in \mathbb{C}^{2^n}$ given by

$$\varphi_T(v) \ := \ T(v) \quad \text{for } v \in \{0,1\}^n.$$

Internal nodes carry no amplitudes; they encode only the binary branching structure. Henceforth, *tree* means perfect binary tree.

Quantum Operations as Tree Transformations. Quantum gates act linearly on the amplitudes of a quantum state. Under the tree representation, this corresponds to local transformations at specific tree levels.

For a node at level k, the two subtrees rooted at $v.0$ and $v.1$ correspond to basis states whose k-th qubit is 0 and 1, respectively. Applying a single-qubit gate to qubit k therefore induces a transformation at level k: for every node v at that level and every suffix w, the amplitudes at leaves $v0w$ and $v1w$ are combined according to the matrix entries of the gate.

Single-qubit gates together with controlled gates are universal for quantum computation (cf. [18]), so it suffices to describe their effect on the tree representation. Consider a controlled-U gate with control qubit c and target qubit k, where U is a single-qubit gate. Such a gate acts only on basis states whose c-th bit is 1. In the tree representation this corresponds to restricting the above transformation to subtrees whose prefix has value 1 at position c. Within those subtrees the amplitudes at leaves $v0w$ and $v1w$ are combined according to U, while the rest of the tree remains unchanged.

3.3.2 LSTAs. We briefly recall the notion of *level-synchronized tree automata (LSTAs)* introduced in [4], in a form specialized to trees defined as above. Let Σ be a ranked alphabet with $\# : \Sigma \to \{0, 2\}$, where symbols of rank 0 label leaves and symbols of rank 2 label internal nodes. We denote the set of rank 0 symbols by Σ_0. An LSTA is a tuple $\mathcal{A} = \langle Q, \Sigma, \Delta, \mathcal{R} \rangle$ where Q is a finite set of *states*, $\mathcal{R} \subseteq Q$ is a set of *root states*, and Δ is a finite set of transitions of the following forms:

$$\begin{aligned} q &\xrightarrow{C} f(q_0, q_1) && \text{internal transition} \\ q &\xrightarrow{C} f && \text{leaf transition} \end{aligned}$$

with $q_0, q_1 \in Q$, $f \in \Sigma$ and $C \subseteq \mathbb{N}_0$ a finite set of *choices*. For a transition $\delta \in \Delta$, we write $\mathtt{top}(\delta)$ for its source state, $\mathtt{sym}(\delta)$ for its symbol, $\ell(\delta)$ for its choice set, and $\mathtt{bot}(\delta)$ for its pair of target states (in the internal case). Transitions with the same source state have pairwise disjoint choice sets, i.e.,

$$\forall \delta_1 \neq \delta_2 \in \Delta : \mathtt{top}(\delta_1) = \mathtt{top}(\delta_2) \implies \ell(\delta_1) \cap \ell(\delta_2) = \emptyset.$$

Semantics. A *run* of $\mathcal{A}$ on a tree $T : \{0,1\}^{\leq n} \to \Sigma$ is a total map $\rho : \{0,1\}^{\leq n} \to \Delta$ such that for every node v, $\mathtt{sym}(\rho(v)) = T(v)$ and, whenever v is internal, $\mathtt{bot}(\rho(v)) = \big(\mathtt{top}(\rho(v.0)), \mathtt{top}(\rho(v.1))\big)$, while v is a leaf node, $\rho(v)$ is of the form $q \xrightarrow{C} T(v)$.

For each *level* d, we define $\mathrm{level}(\rho, d) \coloneqq \{\rho(w) \mid w \in \mathrm{dom}(T) \wedge \mathrm{ht}(w) = d\}$. A run is *level-synchronized* if all transitions from the same level share at least one common choice, or equivalently, for all $d \in \mathbb{N}_0$,

$$C_{d+1} \coloneqq \bigcap_{\delta \in \mathrm{level}(\rho, d)} \ell(\delta) \neq \emptyset.$$

An *accepting* run is a run ρ such that $\mathtt{top}(\rho(\varepsilon)) \in \mathcal{R}$ and ρ is level-synchronized. The *language of* $\mathcal{A}$ is the set $\mathcal{L}(\mathcal{A})$ of trees T with an accepting run.

Notice that an accepting run ρ on a tree of height n determines a sequence

$$(C_1, \ldots, C_{n+1}),$$

We call this the *choice sequence* of ρ and write $\mathrm{choice}(\rho)$. If ρ is an accepting run of $\mathcal{A}$ on a tree T, the sequence $\mathrm{choice}(\rho)$ is also called a *choice sequence of T (with respect to $\mathcal{A}$)*. Different runs may yield different choice sequences.

Returning to Fig. 4, consider the accepting run that generates $|00\rangle$. It uses $r \xrightarrow{\{1\}} x_1(q_1, q_0)$ at the first internal level, $q_1 \xrightarrow{\{2\}} x_2(q_3, q_5)$ together with $q_0 \xrightarrow{\{2,3,4\}} x_2(q_5, q_5)$ at the second, and the

transitions $q_3 \xrightarrow{\{1\}} 1$ and $q_5 \xrightarrow{\{1\}} 0$ at the leaves. The corresponding levelwise intersections are

$$\{1\}, \qquad \{2\} \cap \{2,3,4\} = \{2\}, \qquad \{1\},$$

so the run has choice sequence $(\{1\},\{2\},\{1\})$. In contrast, the earlier modified choice set $\{2,4\}$ at q_0 is incompatible with the choice set $\{3\}$ of the second transition from q_1, since their intersection is empty. The two outgoing transitions from q_1 themselves carry the disjoint sets $\{2\}$ and $\{3\}$, as required for transitions sharing a source state.

4 Quantum Operations and Choice Sequences

We recall how quantum operations are lifted to LSTA. Given an LSTA $\mathcal{A}$ and an operation O —either a quantum gate or a measurement—the constructions in [4, 11] produce an LSTA $O(\mathcal{A})$ that represents the effect of applying O to the trees in $\mathcal{L}(\mathcal{A})$. To ensure a self-contained presentation, we summarize these constructions here for completeness.

The main result of this section concerns the choice sequence. If a tree T is recognized by $\mathcal{A}$ with choice sequence $\mathrm{choice}(\rho_T)$, then the transformed tree $O(T)$ is recognized by $O(\mathcal{A})$ with the same choice sequence. This preservation property does not appear in the works [4, 11] but is essential for the verification framework developed in this paper.

In what follows, we first present the transformation algorithms for unitary gates and measurement, and then prove the preservation theorem.

4.1 Gate Transformation Algorithms

We now describe how quantum operations act on LSTAs. Given an LSTA $\mathcal{A}$ representing a set of quantum states, each operation produces a new LSTA whose language represents the states obtained by applying the corresponding quantum operation to every state in $\mathcal{L}(\mathcal{A})$. The constructions presented here follow the transformation algorithms introduced in [4] and [11].

Single-qubit Gates. Let U_t be a single-qubit unitary U acting on qubit x_t. Applying U_t to an LSTA $\mathcal{A}$ produces a new automaton representing the states obtained by applying U to the t-th qubit of each state recognized by $\mathcal{A}$. Operationally, the transformation acts at level t of the tree and propagates the resulting structure toward the leaves so that the amplitudes are combined according to the matrix entries of U. The precise construction is given in Algorithm 1.

Algorithm 1 Application of a single-qubit gate on an LSTA

Input: An LSTA $\mathcal{A} = \langle Q, \Sigma, \Delta, \mathcal{R}\rangle$ and a gate $\mathrm{U}_t = \left(\begin{smallmatrix} u_1 & u_2 \\ u_3 & u_4 \end{smallmatrix}\right)$
Output: The LSTA $\mathrm{U}_t(\mathcal{A})$

$\Delta'_{<t} \coloneqq \{q \xrightarrow{C} x_i(q_l, q_r) \in \Delta \mid i < t\}$;
$\Delta'_{=t} \coloneqq \{q \xrightarrow{C} x_t((q_l, q_r, L), (q_l, q_r, R)) \mid q \xrightarrow{C} x_t(q_l, q_r) \in \Delta\}$;
$\Delta'_{>t} \coloneqq \{(q_l, q_r, D) \xrightarrow{C_1 \cap C_2} x_i((q_l^l, q_l^r, D), (q_r^l, q_r^r, D)) \mid i > t \wedge q_l \xrightarrow{C_1} x_i(q_l^l, q_r^l), q_r \xrightarrow{C_2} x_i(q_l^r, q_r^r) \in \Delta, D \in \{L, R\}\}$;
$\Delta'_0 \coloneqq \{(q_l, q_r, L) \xrightarrow{C_1 \cap C_2} u_1 \cdot a + u_2 \cdot b, (q_l, q_r, R) \xrightarrow{C_1 \cap C_2} u_3 \cdot a + u_4 \cdot b \mid q_l \xrightarrow{C_1} a, q_r \xrightarrow{C_2} b \in \Delta\}$;
$Q' \coloneqq Q \cup (Q \times Q \times \{L, R\})$; $\Delta' \coloneqq \Delta'_{<t} \cup \Delta'_{=t} \cup \Delta'_{>t} \cup \Delta'_0$; $\Sigma' \coloneqq \Sigma \cup \{u_1 \cdot a + u_2 \cdot b, u_3 \cdot a + u_4 \cdot b \mid a, b \in \Sigma_0\}$;
return $\langle Q', \Sigma', \Delta', \mathcal{R}\rangle$.

Algorithm 1 labels each transition obtained from two original transitions by the intersection $C_1 \cap C_2$, thereby inheriting exactly the synchronization choices common to both. Consider two distinct generated transitions with the same composite source. Their originating pairs differ in at least one component, so the corresponding original transitions from q_l or from q_r are distinct and, by well-formedness of $\mathcal{A}$, have disjoint choice sets. The two generated intersection labels are therefore disjoint as well. Since transitions with original sources retain their original labels, all

outgoing choice sets in $\mathrm{U}_t(\mathcal{A})$ satisfy the pairwise-disjointness condition introduced above, and the resulting automaton is a well-formed LSTA.

Controlled Gates. Controlled operations are handled in a similar manner. Let CU_t^c denote a controlled-U gate with control qubit x_c and target qubit x_t. The transformation modifies the automaton so that the unitary U is applied to the target qubit whenever the control qubit evaluates to 1. The construction preserves the synchronized structure of the automaton while propagating the effect of the gate through the corresponding subtrees. Algorithm 2 presents the complete construction.

Algorithm 2 Application of a controlled gate on an LSTA

Input: An LSTA $\mathcal{A} = \langle Q, \Sigma, \Delta, \mathcal{R}\rangle$, a single-qubit gate $\mathrm{U}_t = \left(\begin{smallmatrix} u_1 & u_2 \\ u_3 & u_4 \end{smallmatrix}\right)$, a control qubit x_c
Output: The LSTA $\mathrm{CU}_t^c(\mathcal{A})$

Build $U_t(\mathcal{A}) = \langle Q^U, \Sigma^U, \Delta^U, \mathcal{R}\rangle$ using Algorithm 1, with $Q^U = Q \cup (Q \times Q \times \{L, R\})$;
Build $\mathcal{A}' = \langle Q', \Sigma, \Delta', \mathcal{R}'\rangle$, a primed copy of $\mathcal{A}$;
for each $\delta = q \xrightarrow{C} x_c(q_1, q_2) \in \Delta^U$ **do**
    if $q_1 \in Q$ **then**
        replace δ with $q \xrightarrow{C} x_c(q_1', q_2)$ in Δ^U;
    else if $q_1 = (q_a, q_b, L)$ **then**
        replace δ with $q \xrightarrow{C} x_c(q_a', q_2)$ in Δ^U;
    else if $q_1 = (q_a, q_b, R)$ **then**
        replace δ with $q \xrightarrow{C} x_c(q_b', q_2)$ in Δ^U;
return $\langle Q^U \cup Q', \Sigma^U \cup \Sigma, \Delta^U \cup \Delta', \mathcal{R}\rangle$

Measurement. Measurements require a slightly different construction. Given a target qubit x_i and a measurement outcome $b \in \{0, 1\}$, the transformation restricts the amplitudes of the states represented by the automaton to those consistent with the observed outcome. Algorithm 3 performs this construction.

Algorithm 3 Application of measurement

Input: An LSTA $\mathcal{A} = \langle Q, \Sigma, \Delta, \mathcal{R}\rangle$, target qubit x_i, measurement outcome b
Output: The LSTA $M_i^{=b}(\mathcal{A})$

$Q' := \{q' \mid q \in Q\}$;
$\Delta' := \{q' \xrightarrow{C} x(q_1', q_2') \mid q \xrightarrow{C} x(q_1, q_2) \in \Delta\} \cup \{q' \xrightarrow{C} 0 \mid q \xrightarrow{C} a \in \Delta\}$;
$\Delta^{\neq x_i} := \{q \xrightarrow{C} x(q_1, q_2) \mid q \xrightarrow{C} x(q_1, q_2) \in \Delta \wedge x \neq x_i\} \cup \{q \xrightarrow{C} a \mid q \xrightarrow{C} a \in \Delta\}$;
if $b = 0$ **then**
    $\Delta^{=x_i} := \{q \xrightarrow{C} x_i(q_0, q_1') \mid q \xrightarrow{C} x_i(q_0, q_1) \in \Delta\}$;
else
    $\Delta^{=x_i} := \{q \xrightarrow{C} x_i(q_0', q_1) \mid q \xrightarrow{C} x_i(q_0, q_1) \in \Delta\}$;
return $M_i^{=b}(\mathcal{A}) = \langle Q \cup Q', \Sigma \cup \{0\}, \Delta^{=x_i} \cup \Delta^{\neq x_i} \cup \Delta', \mathcal{R}\rangle$;

The measurement transformation does not perform the normalization step that usually follows a quantum measurement. Instead amplitudes associated with non-observed outcomes are set to zero while the remaining amplitudes are left unchanged. Performing normalization at the level of automata would require computing a global scaling factor for each accepted tree, which would considerably complicate the construction.

Rather than introducing such normalization into the automaton transformations, we reason about quantum states up to a nonzero scalar factor. This issue will be addressed later by the up-to-scale variant of choice-aware language inclusion introduced in Section 5.3.

4.2 Choice-Sequence Preservation

The transformation algorithms of the previous subsection construct the automaton $O(\mathcal{A})$ by combining transitions of $\mathcal{A}$ according to the semantics of the quantum operation O. The formal correctness of these constructions is established in [1], which proves a strict bijection between the accepting runs of the input automaton $\mathcal{A}$ and the accepting runs of the output automaton $O(\mathcal{A})$. Building upon this structural equivalence, we establish that these transformations also preserve the choice sequence of the runs. In these constructions, the choice sets attached to new transitions are obtained from those of the original transitions, typically by taking their intersections. Consequently, the choice information carried by a run is propagated through the construction in a way that respects the level-synchronization discipline.

Intuitively, if a run of $\mathcal{A}$ selects transitions whose choice sets intersect to form the sequence $(C_1, \ldots, C_{n+1})$, then the corresponding run constructed in $O(\mathcal{A})$ selects transitions whose choice sets yield the same intersections at each level. The choice sequence is therefore preserved. The following theorem makes this statement precise.

Theorem 4.1 (Choice-sequence preservation). *Let $\mathcal{A}$ be an LSTA and let O be a quantum operation. For every $T \in \mathcal{L}(\mathcal{A})$ and every accepting run ρ of $\mathcal{A}$ on T with choice sequence* $\mathrm{choice}(\rho) = (C_1, \ldots, C_{n+1})$*, there exists an accepting run ρ' of $O(\mathcal{A})$ on the transformed tree $O(T)$ such that* $\mathrm{choice}(\rho') = \mathrm{choice}(\rho)$.

Proof. Let $T \in \mathcal{L}(\mathcal{A})$ and let ρ be an accepting run of $\mathcal{A}$ on T with choice sequence

$$(C_1, \ldots, C_{n+1}).$$

By definition of acceptance, ρ is level-synchronized and $\mathtt{top}(\rho(\varepsilon)) \in \mathcal{R}$. We construct a run ρ' of $U_t(\mathcal{A})$ on the transformed tree $U_t(T)$ by following the structure of Algorithm 1. The construction proceeds level by level.

For levels $i < t$, the algorithm copies the transitions of $\mathcal{A}$ unchanged (Line 1). Hence, we set $\rho'(v) = \rho(v)$ for all nodes v with $|v| < t$. The choice sets at these levels are therefore identical, and the components C_i of the choice sequence remain unchanged.

At level t, each transition

$$q \xrightarrow{C} x_t(q_l, q_r)$$

is replaced by a transition whose children states encode the pair of successor states together with a direction tag $D \in \{L, R\}$ (Line 2). The choice set C is preserved. Thus the intersections of choice sets at level t remain equal to C_t.

For levels $i > t$, the algorithm combines pairs of transitions of $\mathcal{A}$ (Lines 3). If ρ uses the transitions

$$q_l \xrightarrow{C^1} x_i(q_l^l, q_r^l) \quad \text{and} \quad q_r \xrightarrow{C^2} x_i(q_l^r, q_r^r),$$

then ρ' uses the transition constructed from this pair, whose choice set is $C^1 \cap C^2$. Because ρ is level-synchronized, the intersections of the choice sets of all transitions used at level i contain C_i. Consequently, the transitions used by ρ' at the same level have choice sets whose intersection is still C_i.

Finally, at the leaves the amplitudes are combined according to the matrix entries of U (Lines 4). The corresponding transitions again use the choice sets $C^1 \cap C^2$, so the intersection property is preserved.

Thus the constructed run ρ' is level-synchronized and satisfies $\text{choice}(\rho') = \text{choice}(\rho)$. Since the root state condition is unchanged, ρ' is an accepting run of $U(\mathcal{A})$ on $U(T)$.

The proofs for controlled-U gates (Algorithm 2) and measurement (Algorithm 3) follow the same fashion. □

The preservation theorem establishes that the choice sequence of an accepting run is invariant under the quantum operations considered in this work. Although the transformation algorithms introduce new states and transitions in the automaton, the synchronization discipline ensures that the intersections defining the choice sequence remain unchanged. Consequently, each tree produced by executing a quantum operation on a tree recognized by $\mathcal{A}$ can be associated with a run of the transformed automaton that carries exactly the same choice sequence.

This invariant plays a central role in the verification framework developed in this paper. The choice sequence serves as a structural identifier of executions: when a quantum program is applied to a precondition automaton, the resulting automaton represents the set of output states while preserving the choice sequence of each run. Thus, every run of the resulting automaton corresponds to a run of the precondition automaton with the same choice sequence.

The transformations therefore preserve the same choice sequence for corresponding runs. Ordinary language inclusion compares only the recognized trees and does not retain this run correspondence. Choice-aware language inclusion instead compares runs with compatible choice sequences; the next section formalizes this relation and presents an algorithm for deciding it.

5 Choice-Aware Language Inclusion and Verification Framework

We now describe how LSTAs can be used to verify functional correctness of quantum programs.

In our framework, set of quantum states are represented by LSTAs. Given an LSTA $\mathcal{A}_{\text{pre}}$ describing the input states of a program and an LSTA $\mathcal{A}_{\text{post}}$ describing the intended output states, program correctness requires that every input state be transformed into the corresponding output state.

Choice sequences play a central role in establishing this correspondence. As shown in Theorem 4.1 of the previous section, the quantum operations considered in this work preserve the choice sequence associated with a run of LSTA. Consequently, when a program is applied to an LSTA representing input states, the resulting LSTA represents the corresponding output states while maintaining the same choice sequences. These sequences therefore provide a natural index that relates input states with their outputs.

Using this observation, we first introduce a specification framework that expresses functional correctness of quantum programs in terms of LSTAs. We then show that verifying such specifications reduces to a *language inclusion problem between LSTAs that takes choice sequences into account*. Finally, we develop algorithms for deciding this relation and extend them to an *up-to-scale* variant required for reasoning about quantum programs involving measurements and global phase differences.

5.1 Functional Correctness Specification

We specify the functional behavior of a quantum program using LSTAs that represent sets of quantum states. Let P be a quantum program, and let $\mathcal{A}_{\text{pre}}$ and $\mathcal{A}_{\text{post}}$ be LSTAs representing the sets of input and output states, respectively. Here, the precondition automaton $\mathcal{A}_{\text{pre}}$ is chosen so that its accepted trees have distinct choice sequences. This causes no loss of generality: a set of quantum states may be represented by many different LSTAs, and runs with different choice sequences can be separated by a routine refinement of the automaton. Hence, each input state represented by $\mathcal{A}_{\text{pre}}$ is associated, in this representation, with a unique choice sequence. The postcondition LSTA $\mathcal{A}_{\text{post}}$ is specified accordingly. For each such sequence it recognizes the state that should result

from executing the program on the corresponding input. Inputs and outputs are therefore matched by equality of their choice sequences.

Executing the program P on $\mathcal{A}_{\text{pre}}$ yields an LSTA $P(\mathcal{A}_{\text{pre}})$ obtained by applying the quantum operations of P to $\mathcal{A}_{\text{pre}}$. By the preservation Theorem 4.1 of Section 4, these operations preserve the choice sequence of each accepting run. Thus, the same sequence identifies both the input state and the output state produced from it by the program.

This observation leads to the following notion of *functional correctness*. The specification

$$\{\!|\mathcal{A}_{\text{pre}}|\!\}\ \ P\ \ \{\!|\mathcal{A}_{\text{post}}|\!\} \tag{4}$$

is said to hold if every tree produced by executing P on an input state represented by $\mathcal{A}_{\text{pre}}$ must be accepted by $\mathcal{A}_{\text{post}}$ with the same choice sequence. Such a specification will be referred to as a *functional specification* or a *functional triple*.

Verifying a functional specification therefore reduces to checking a language inclusion relation between LSTAs that takes choice sequences into account.

5.2 Choice-Aware Language Inclusion

Verifying a functional specification of a program reduces to checking whether the LSTA produced by executing the program satisfies the postcondition LSTA. As explained in the previous subsection, this requires comparing LSTAs not only by the trees they recognize but also by the choice sequences associated with those trees.

Let $\mathcal{A}$ and $\mathcal{B}$ be LSTAs. For a tree $T \in \mathcal{L}(\mathcal{A})$, let ρ denote an accepting run of $\mathcal{A}$ on T and let $\text{choice}(\rho)$ denote its choice sequence. We write

$$\mathcal{L}(\mathcal{A}) \subseteq_{\text{choice}} \mathcal{L}(\mathcal{B}) \tag{5}$$

if for every tree $T \in \mathcal{L}(\mathcal{A})$ with accepting run ρ in $\mathcal{A}$, there exists an accepting run ρ' of $\mathcal{B}$ on the same tree T such that their choice sequences are *compatible*. That is, if $\text{choice}(\rho) = (C_1, \ldots, C_{n+1})$ and $\text{choice}(\rho') = (C'_1, \ldots, C'_{n+1})$, then $C_d \cap C'_d \neq \emptyset$ for all $1 \leq d \leq n+1$. We refer to this relation as *choice-aware language inclusion*. Thus the inclusion relation compares trees together with their associated choice sequences.

We now describe how this relation can be decided. The key observation is that LSTAs are level-synchronized. Consequently the choice sequence of a run is determined level by level. Runs of two LSTAs can therefore be explored simultaneously while requiring their levelwise choice sets to be compatible, that is, to share at least one choice at each level. The decision procedure is presented in Algorithm 4, which supports Theorem 5.2. The algorithm employs a worklist-based approach to explore the synchronized runs of $\mathcal{A}$ and $\mathcal{B}$. Each element in the *workset* is a simulation configuration (D, F), where D is the current state frontier of one source run being explored in $\mathcal{A}$, and F stores the target configurations in $\mathcal{B}$ that can still match that source-run prefix. Specifically, F consists of pairs (f, g), where f maps each source-frontier state in D to candidate matching states in $\mathcal{B}$, while g records the corresponding source and target leaf symbols accumulated so far.

Starting from the root states (Line 1), the procedure systematically advances level by level. We use the function $\texttt{FeasTrans}(\mathcal{M}, Q_{\mathcal{M}})$ for an arbitrary LSTA $\mathcal{M}$ and a set of states $Q_{\mathcal{M}}$ to denote a collection of sets of *synchronized* transitions. Each transition set contains exactly one transition δ_s going downward from each $s \in Q_{\mathcal{M}}$ (i.e., $\texttt{top}(\delta_s) = s$). For each valid set of synchronized transitions $\Gamma_{\mathcal{A}}$ from D, it computes the next frontier D' for $\mathcal{A}$ (Lines 8–9). Simultaneously, it explores all feasible transition sets $\Gamma_{\mathcal{B}}$ in $\mathcal{B}$ from the mapped states $\text{img}(f)$ (Line 11). To maintain choice compatibility, the algorithm discards a pair of transition sets $\Gamma_{\mathcal{A}}$ and $\Gamma_{\mathcal{B}}$ when their joint choice-set intersection is empty (Line 12). This is the step that enforces choice-awareness during the exploration.

As the algorithm traverses the transitions (Lines 14–21), it constructs the next-level mappings f' and g'. If the transitions are internal, the algorithm propagates the matching positionally: f' matches left children only with left children and right children only with right children (Line 19). If the transitions are leaves, the corresponding source and target symbols are recorded in g' (Line 17).

The exploration reaches the bottom of the trees when the frontier becomes empty, i.e., $D = \emptyset$ (Line 4). At this point, the algorithm must verify whether the leaves of trees generated by the current run of $\mathcal{A}$ can be matched by at least one valid run in $\mathcal{B}$. It evaluates the accumulated information in F: it checks whether F contains at least one candidate $(\emptyset, g)$ for which every recorded source leaf symbol equals its corresponding target leaf symbol (Line 5). If no such candidate exists, the explored source run has no compatible target run on the same tree, and the procedure returns false (Line 6). If the worklist is exhausted without finding any counterexample, the inclusion holds, and the algorithm returns true (Line 26).

Algorithm 4 Checking if $\mathcal{L}(\mathcal{A}) \subseteq_{\text{choice}} \mathcal{L}(\mathcal{B})$

Input: LSTAs $\mathcal{A} = \langle Q_{\mathcal{A}}, \Sigma, \Delta_{\mathcal{A}}, \mathcal{R}_{\mathcal{A}} \rangle$ and $\mathcal{B} = \langle Q_{\mathcal{B}}, \Sigma, \Delta_{\mathcal{B}}, \mathcal{R}_{\mathcal{B}} = \{r_1, \ldots, r_k\} \rangle$
Output: **true** if $\mathcal{L}(\mathcal{A}) \subseteq_{\text{choice}} \mathcal{L}(\mathcal{B})$, **false** otherwise

1: $workset \leftarrow \{(\{q\}, \{(\{q \mapsto \{r_1\}\}, \emptyset), \ldots, (\{q \mapsto \{r_k\}\}, \emptyset)\}) \mid q \in \mathcal{R}_{\mathcal{A}}\}$; $processed \leftarrow \emptyset$;
2: **while** $\exists (D, F) \in workset$ **do**
3: $workset$.remove((D, F)); $processed$.insert((D, F));
4: **if** $D = \emptyset$ **then**
5: **if** $\neg \bigvee_{(\emptyset, g) \in F} \bigwedge_{\substack{(u_1 \mapsto U_2) \in g \\ u_2 \in U_2}} u_1 = u_2$ **then**
6: **return false**;
7: **else**
8: **for** each $\Gamma_{\mathcal{A}} \in \texttt{FeasTrans}(\mathcal{A}, D)$ **do**
9: $D' \leftarrow \{q \in \texttt{bot}(\delta) \mid \delta \in \Gamma_{\mathcal{A}}\}$; $F' \leftarrow \emptyset$;
10: **for** each $(f, g) \in F$ **do**
11: **for** each $\Gamma_{\mathcal{B}} \in \texttt{FeasTrans}(\mathcal{B}, \text{img}(f))$ **do**
12: **if** $\bigcap_{\delta \in \Gamma_{\mathcal{A}} \cup \Gamma_{\mathcal{B}}} \ell(\delta) = \emptyset$ **then continue**;
13: $f' \leftarrow \emptyset$; $g' \leftarrow \emptyset$; $failed \leftarrow$ **false**;
14: **for** each $\delta_{\mathcal{A}} \in \Gamma_{\mathcal{A}}$ and $q \in f(\texttt{top}(\delta_{\mathcal{A}}))$ **do**
15: $\{\delta_{\mathcal{B}}\} \leftarrow \{\delta_{\mathcal{B}} \in \Gamma_{\mathcal{B}} \mid q = \texttt{top}(\delta_{\mathcal{B}})\}$;
16: **if** $a = \texttt{sym}(\delta_{\mathcal{A}})$ and $b = \texttt{sym}(\delta_{\mathcal{B}})$ are both leaf symbols **then**
17: $g'(a)$.insert(b);
18: **else if** $\texttt{sym}(\delta_{\mathcal{A}}) = \texttt{sym}(\delta_{\mathcal{B}})$ is an internal symbol **then**
19: $f'(\texttt{left}(\delta_{\mathcal{A}}))$.insert($\texttt{left}(\delta_{\mathcal{B}})$); $f'(\texttt{right}(\delta_{\mathcal{A}}))$.insert($\texttt{right}(\delta_{\mathcal{B}})$);
20: **else**
21: $failed \leftarrow$ **true**; **break**;
22: **if** $\neg failed$ **then**
23: F'.insert($(f', g \uplus g')$);
24: **if** $(D', F') \notin processed \cup workset$ **then**
25: $workset$.insert((D', F'));
26: **return true**;

Lemma 5.1. *Algorithm 4 always terminates and returns true if and only if* $\mathcal{L}(\mathcal{A}) \subseteq_{\text{choice}} \mathcal{L}(\mathcal{B})$.

The proof of Lemma 5.1 consists of two main parts: termination and correctness. First, termination is guaranteed because the state spaces $Q_{\mathcal{A}}$ and $Q_{\mathcal{B}}$, as well as the alphabet Σ, are finite. Consequently,

the number of distinct configurations (D, F) constructed by the algorithm is strictly bounded. Since the procedure maintains a *processed* set to prevent revisiting any previously evaluated configuration, the *workset* will eventually be exhausted, ensuring the algorithm terminates in finite time.

For correctness, we establish a structural invariant based on the level-by-level exploration of the perfect binary trees. We show that for each accepting run produced by $\mathcal{A}$, the algorithm computes a uniquely determined sequence of configurations $(D_0, F_0), (D_1, F_1), \ldots, (\emptyset, F_{n+1})$. Each pair (D_i, F_i) guarantees that F_i precisely tracks all valid, choice-synchronized subruns in $\mathcal{B}$ that structurally match the $\mathcal{A}$-run up to depth i. When the exploration reaches the leaves ($D = \emptyset$), the accumulator g within F_n holds the complete pairing of generated leaf symbols. The evaluation condition (Line 5) then directly decides if at least one tracked $\mathcal{B}$-run perfectly matches these symbols, successfully determining whether the $\mathcal{A}$-run can be entirely covered by $\mathcal{B}$. Lemma 5.1 directly establishes the following theorem.

Theorem 5.2. *The choice-aware language inclusion between LSTAs is decidable.*

5.3 Up-to-Scale Inclusion

As discussed in Section 4.1, the measurement transformation does not normalize the resulting quantum state. Instead amplitudes associated with non-observed outcomes are set to zero while the remaining amplitudes are left unchanged. Consequently the trees produced by the measurement operator represent vectors that may differ from normalized quantum states by a nonzero scalar factor.

Because nonzero scalar multiples normalize to the same quantum state, verification should compare these vectors up to proportionality rather than by exact equality. Recall from Section 3 that the leaves of a tree store complex amplitudes and therefore represent a vector in $\mathbb{C}^{2^n}$. Such vectors that differ only by a nonzero scalar factor represent the same quantum state up to global scaling.

We therefore extend the notion of choice-aware language inclusion introduced in the previous subsection to an *up-to-scale* variant. Let T_1 and T_2 be trees of the same height. We write

$$T_1 \sim T_2$$

if there exists a nonzero scalar $\lambda \in \mathbb{C}$ such that the vector represented by T_1 equals λ times the vector represented by T_2.

Using this relation we define the following extension of choice-aware language inclusion. Let $\mathcal{A}$ and $\mathcal{B}$ be LSTAs. We write

$$\mathcal{L}(\mathcal{A}) \subseteq^{\text{uts}}_{\text{choice}} \mathcal{L}(\mathcal{B})$$

if for every tree $T \in \mathcal{L}(\mathcal{A})$ with accepting run ρ in $\mathcal{A}$, there exist a tree $T' \in \mathcal{L}(\mathcal{B})$ and an accepting run ρ' of $\mathcal{B}$ on T' such that

$$T \sim T' \qquad \text{and} \qquad C_d \cap C'_d \neq \emptyset \quad \text{for all } 1 \leq d \leq n+1, \tag{6}$$

where $\text{choice}(\rho) = (C_1, \ldots, C_{n+1})$ and $\text{choice}(\rho') = (C'_1, \ldots, C'_{n+1})$. Thus the inclusion relation compares trees up to a scalar factor while still requiring their choice sequences to be compatible.

The decision procedure for this relation is a straightforward extension of the Algorithm 4 for choice-aware language inclusion described in Section 5.2. The only difference is that the amplitudes computed at the leaves are compared up to scale rather than by exact equality. This can be done by modifying the condition in Line 5 of Algorithm 4 to $\neg \bigvee_{(\emptyset,g)\in F} \exists \lambda \in \mathbb{C} \setminus \{0\} : \bigwedge_{\substack{(u_1 \mapsto U_2)\in g \\ u_2 \in U_2}} u_1 = \lambda \cdot u_2$.

Theorem 5.3. *Choice-aware inclusion up to scale between LSTAs is decidable.*

Summary of the Verification Framework. This section introduced the verification framework used in this work. Functional correctness of a quantum program is specified using precondition and postcondition automata together with the correspondence induced by choice sequences. Verification then reduces to checking choice-aware language inclusion between the automaton obtained by executing the program and the postcondition automaton. When measurements are involved, the comparison is performed up to scale using the relation $\subseteq^{\text{uts}}_{\text{choice}}$ introduced above.

6 Verifying Repeat-until-Success Protocol

In this section we show how RUS protocols can be verified within the automata-based framework developed in the previous sections. The presentation proceeds in two steps. First, we establish a characterization theorem showing that a candidate circuit implements a correct RUS step if it behaves properly on three carefully chosen test states. Second, we explain how this characterization enables the verification of the entire protocol using the functional specification framework of Section 5.

6.1 RUS Certification

We consider the problem of verifying a 2-qubit RUS implementation of a single-qubit unitary U. Recall that an RUS *protocol* repeatedly executes an RUS circuit and measures an ancilla qubit until the measurement outcome indicates success. The protocol operates on two qubits: an ancilla qubit and a data qubit. The ancilla is initialized to $|0\rangle$ at the beginning of each attempt. Let $F : \mathbb{C}^2 \otimes \mathbb{C}^2 \to \mathbb{C}^2 \otimes \mathbb{C}^2$ denote the linear operator implemented by the unitary portion of the RUS before the measurement, where the first qubit corresponds to the ancilla. One attempt of the protocol therefore consists of applying F, measuring the ancilla, and branching on the measurement result.

For the protocol to implement the target unitary U, the circuit F must have a very particular structure. We say that F has *RUS form* for U if there exist a single-qubit unitary R and complex constants a, b such that for every data state $|\varphi\rangle$,

$$F(|0\rangle\,|\varphi\rangle) = a\ |1\rangle\ U\,|\varphi\rangle + b\ |0\rangle\ R\,|\varphi\rangle\,. \tag{7}$$

In this situation a measurement of the ancilla yields outcome 1 with probability $|a|^2$ and produces the state $U\,|\varphi\rangle$. If the outcome is 0, the state collapses to $R\,|\varphi\rangle$. Applying the repair operation $R^\dagger$ restores the data qubit to $|\varphi\rangle$, returning the system to the configuration $|0\rangle\,|\varphi\rangle$ for the next attempt.

The following theorem shows that this structural property of F can be certified by testing the circuit on only three input states.

Theorem 6.1 (Three-test certification). *Let U be the target single-qubit unitary and let R be the repair unitary applied after an unsuccessful attempt of the RUS protocol implementing U. Let*

$$F : \mathbb{C}^2 \otimes \mathbb{C}^2 \to \mathbb{C}^2 \otimes \mathbb{C}^2$$

be the linear operator implemented by one attempt of the protocol before the ancilla measurement. For each $\varphi \in \{0, 1, +\}$ denote by $a_\varphi, b_\varphi \in \mathbb{C}$ coefficients such that

$$F(|0\rangle\,|\varphi\rangle) = a_\varphi\ |1\rangle\ U\,|\varphi\rangle + b_\varphi\ |0\rangle\ R\,|\varphi\rangle\,.$$

Then there exist constants $a, b \in \mathbb{C}$, independent of the input state, such that

$$\forall\ |\varphi\rangle \in \mathbb{C}^2 : \qquad F(|0\rangle\,|\varphi\rangle) = a\ |1\rangle\ U\,|\varphi\rangle + b\ |0\rangle\ R\,|\varphi\rangle$$

Intuitively, verifying the above relation on the basis states $|0\rangle$ and $|1\rangle$ alone is not sufficient. These two tests determine the action of F on basis vectors but still allow the coefficients of the success and failure branches to depend on the basis direction.

We first illustrate that testing only on the basis states $|0\rangle$ and $|1\rangle$ is insufficient. Consider a faulty RUS circuit F' intended to implement the identity $U = I$ with $R = I$. Suppose F' behaves on the basis states as follows:

$$F'(|0\rangle|0\rangle) = \frac{1}{\sqrt{2}}|1\rangle|0\rangle + \frac{1}{\sqrt{2}}|0\rangle|0\rangle \quad \text{and} \quad F'(|0\rangle|1\rangle) = -\frac{1}{\sqrt{2}}|1\rangle|1\rangle + \frac{1}{\sqrt{2}}|0\rangle|1\rangle$$

If we only test the computational basis, the success branch (ancilla $|1\rangle$) yields $|0\rangle$ for input $|0\rangle$, and (up to a global phase) $|1\rangle$ for input $|1\rangle$. Both appear correct in isolation. However, by linearity, evaluating the circuit on the superposition state $|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ yields:

$$F'(|0\rangle|+\rangle) = \frac{1}{2}|1\rangle(|0\rangle - |1\rangle) + \frac{1}{2}|0\rangle(|0\rangle + |1\rangle)$$

The data qubit collapses to $|-\rangle$ in the success branch, which is strictly incorrect since $U|+\rangle = |+\rangle$. The third test state $|+\rangle$ explicitly enforces coefficient consistency ($a_0 = a_1$) across branches, preventing such relative phase errors.

Proof. Using $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ and linearity of F we obtain

$$F(|0\rangle|+\rangle) = \frac{1}{\sqrt{2}}\big(F(|0\rangle|0\rangle) + F(|0\rangle|1\rangle)\big).$$

Substituting the assumed expressions for the three test cases yields

$$a_+U|+\rangle = \frac{1}{\sqrt{2}}(a_0U|0\rangle + a_1U|1\rangle) \quad \text{and} \quad b_+R|+\rangle = \frac{1}{\sqrt{2}}(b_0R|0\rangle + b_1R|1\rangle).$$

Since U and R are unitary, the vectors $\{U|0\rangle, U|1\rangle\}$ and $\{R|0\rangle, R|1\rangle\}$ form bases. Comparing coefficients in these bases gives

$$a_0 = a_1 = a_+, \qquad b_0 = b_1 = b_+.$$

Let $a = a_0$ and $b = b_0$. Then the desired relation holds for every $|\varphi\rangle$ by linearity. □

Generalization to m-ancilla and n-target Qubits. While Theorem 6.1 focuses on the standard two-qubit RUS setting, the proof structure naturally generalizes. For an n-qubit target unitary, a $(2^n + 1)$-test theorem applies: testing the 2^n computational basis states determines the action on each orthogonal direction, while an additional superposition test (e.g., $|+\rangle^{\otimes n}$) enforces global coefficient consistency via linearity. Practically, this $(2^n + 1)$ generalization is well-suited for our framework, since LSTAs encode the computational basis with a size that scales linearly with n. Furthermore, generalizing to m-ancilla qubits expands the linear combination in Eq. (7) to include one success branch and $2^m - 1$ distinct failure branches. The identical superposition argument ensures coefficient consistency across all these branches simultaneously.

The theorem shows that the structure of a valid RUS step can be determined from the behaviour of the circuit on only three input states.

6.2 Verification of the RUS Protocol

We now explain how this certification result is used in the verification framework of Section 5. The protocol may be written explicitly as the following while-loop program RUS:

$$F; \texttt{measure ancilla}; \ \textbf{while}\ b = 0\ \textbf{do}\ \{R^\dagger; F; \texttt{measure ancilla}\}.$$

Here the variable b records the measurement outcome of the ancilla. The desired property is a functional specification

$$\wr\mathcal{A}_{\text{pre}}\wr \ \text{RUS} \ \wr\mathcal{A}_{\text{post}}\wr$$

stating that whenever the protocol terminates, the data qubit contains the state $U|\varphi\rangle$ corresponding to the input $|\varphi\rangle$. This establishes *functional correctness* of the protocol.

At first sight this specification requires reasoning about all possible two-qubit input states of the form $|0\varphi\rangle$. Three-test theorem Theorem 6.1 provides a crucial simplification. It shows that the behaviour of the circuit F is completely determined by its action on the three states $|00\rangle, |01\rangle$ and $|0+\rangle$. We therefore represent the precondition by an LSTA $\mathcal{A}_{\text{pre}}$ recognizing $\{|00\rangle, |01\rangle, |0+\rangle\}$, and the postcondition by an LSTA $\mathcal{A}_{\text{post}}$ recognizing $\{|1\rangle U|0\rangle, |1\rangle U|1\rangle, |1\rangle U|+\rangle\}$.

This reduction is not merely conceptual but also important computationally. Without the Theorem 6.1 one would need to represent symbolic two-qubit states using variables. The resulting up-to-scale inclusion checks would generate SMT queries containing mixtures of quantified nonlinear integer arithmetic and nonlinear real arithmetic constraints. Such formulas are undecidable in general and difficult for SMT solvers to handle in practice. Restricting the verification to the three test states keeps the generated constraints within decidable fragments that SMT solvers can solve efficiently.

Since the RUS protocol is a while-loop program, its verification follows the usual invariant reasoning for loops. Let $\mathcal{I}$ denote the loop invariant. To establish the functional specification it suffices to check the following three triples:

$$\{\!|\mathcal{A}_{\text{pre}}|\!\}\ F\ \{\!|\mathcal{I}|\!\}, \qquad \{\!|M_1^{=0}(\mathcal{I})|\!\}\ R^\dagger; F\ \{\!|\mathcal{I}|\!\}, \qquad \{\!|M_1^{=1}(\mathcal{I})|\!\}\ \texttt{skip}\ \{\!|\mathcal{A}_{\text{post}}|\!\}.$$

The first triple ensures that all states reachable before entering the loop satisfy the invariant. The second triple establishes that the invariant is preserved across loop iterations when the measurement outcome is 0. The third triple guarantees that when the loop terminates (i.e., the measurement outcome is 1), the resulting state satisfies the postcondition.

A notable feature of our framework is that the required invariant does not need to be supplied by the user. Instead it can be synthesized automatically from the structure of the RUS protocol. Recall that each loop iteration begins by applying the circuit F to a state of the form $|0\rangle|\varphi\rangle$. After a failing outcome, the repair operation restores the data qubit to its original state, so that the next attempt again starts from a precondition state. Consequently, the states obtained immediately after applying F to the precondition states characterize all configurations that arise before each measurement. This observation suggests the invariant

$$\mathcal{I} := F(\mathcal{A}_{\text{pre}}),$$

obtained by applying the gate-transformation algorithms of Section 4.1 to the precondition automaton.

By construction, $\mathcal{I}$ represents precisely the states reachable after one application of the circuit and before the measurement, and these are exactly the states at the beginning of every loop iteration. Moreover, because the invariant is defined as $\mathcal{I} = F(\mathcal{A}_{\text{pre}})$, the initialization condition $\{\!|\mathcal{A}_{\text{pre}}|\!\}\ F\ \{\!|\mathcal{I}|\!\}$ holds by construction. The first verification triple is therefore redundant and does not need to be checked explicitly.

The verification task thus reduces to checking the remaining two triples. The first of these establishes that the invariant is preserved when the measurement outcome is 0, i.e. when the loop continues. The second ensures that when the loop terminates (when the measurement outcome is 1), the resulting state satisfies the postcondition.

Both conditions are discharged automatically using the choice-aware up-to-scale language inclusion relation introduced in Section 5. Intuitively, these checks verify that the trees representing the reachable quantum states remain within the invariant after a failing iteration, and that the terminating states belong to the postcondition.

Because the automata transformations preserve choice sequences and choice-aware inclusion requires compatible choice sequences between the compared runs, the verification establishes not only that the postcondition holds but also that each input state is matched with the corresponding output state produced by the protocol. This correspondence is essential for establishing functional correctness rather than merely verifying state-set properties.

Scope of Invariant Synthesis. It is important to note that our automatic loop invariant synthesis, $I = F(\mathcal{A}_{pre})$, relies on a specific structural property of the RUS protocols considered here: all failure branches are actively repaired to restore the initial precondition state before the next iteration begins. While our automata-based structural design handles probabilistic branching and unbounded looping robustly, we do not claim that this automatic synthesis method generalizes to arbitrary adaptive quantum programs or general while-loops that lack such self-restoring configurations.

Generalization to m-ancilla Branches. When extending the verification framework to m ancilla qubits, the protocol yields $2^m - 1$ distinct failure branches. In our framework, this requires checking the invariant preservation triple $\{\!|M^{=i}(\mathcal{I})|\!\}\ R_i^\dagger; F\ \{\!|\mathcal{I}|\!\}$ for each failure outcome i. Since quantum operations and measurements preserve choice sequences (Theorem 4.1), the choice-aware language inclusion handles each branch independently.

Overall, the verification of the RUS protocol reduces to automata transformations together with the corresponding language inclusion checks. Both steps can be carried out automatically by our algorithms, allowing the functional correctness of the protocol to be certified without user-provided invariants or manual reasoning.

7 Evaluation Results

We implemented the verification framework described in the previous sections and evaluated it on a collection of repeat-until-success (RUS) protocols drawn from the literature [19]. The experiments were conducted on a laptop with Intel Core i7-12700H as CPU equipped with 64 GB RAM and running Ubuntu 24.04 as the operation system.

We first verified a number of RUS circuits proposed in [19]. Each program implements a single-qubit unitary using a two-qubit circuit with an ancilla. For each circuit we checked the functional specification introduced in Section 5.1. Table 1 summarizes the results. The table lists the target unitary, the number of qubits and gates in the program, and the verification outcome. Our tool successfully verified all correct implementations and detected two erroneous ones in [19]. For each incorrect circuit the table also shows the corrected version, which our verifier confirms to be correct.

These results illustrate two aspects of the framework. First, the verifier is able to detect subtle mistakes in program implementations. Second, all evaluated instances completed within 116 ms, with reported memory usage 27 MB.

To evaluate the compositional capabilities of the framework, we constructed programs by composing pairs of RUS protocols. Each composed program executes two RUS circuits sequentially and therefore contains a larger circuit and requires two program invariants. Table 2 reports the results. The framework successfully verifies the correctness of all valid compositions. Moreover, when a composition involves a buggy RUS protocol, the verifier correctly detects the error. The reported running time range from 106 ms to 167 ms across these evaluated instances, while the reported memory usage ranges from 26 MB to 27 MB. This demonstrates that the approach remains effective when protocols are combined into larger programs.

Across both experiments, all evaluated benchmarks completed within 167 ms, with reported memory usage ranging from 26 MB to 27 MB. The nearly constant memory usage is consistent with

Table 1. Results of verifying *Repeat-until-Success* examples from [19]

Repeat-until-Success [19]					
program	**qubits**	**gates**	**result**	**time**	**memory**
$V_7 = (2X + \sqrt{2}Y + Z)/\sqrt{7}$	2	30	OK	116ms	27MB
$V_8 = (I + i\sqrt{2}X)/\sqrt{3}$	2	17	OK	84ms	27MB
$V_9 = (I + 2iZ)/\sqrt{5}$	2	27	OK	89ms	27MB
$V_{10a-old} = (3I + 2iZ)/\sqrt{13}$	2	43	failed	93ms	25MB
$V_{10a} = (3I - 2iZ)/\sqrt{13}$ (corrected)	2	43	OK	97ms	27MB
$V_{10b} = (4I + iZ)/\sqrt{17}$	2	76	OK	104ms	27MB
$V_{10c-old} = (5I + 2iZ)/\sqrt{29}$	2	67	failed	96ms	27MB
$V_{10c} = (5I - 2iZ)/\sqrt{29}$ (corrected)	2	67	OK	95ms	27MB

Table 2. Results of verifying compositions of RUS from Table 1

Sequential compositions of Repeat-until-Success from [19]					
program	**qubits**	**gates**	**result**	**time**	**memory**
$V_7 \circ V_7$	2	61	OK	121ms	27MB
$V_7 \circ V_8$	2	48	OK	167ms	26MB
$V_8 \circ V_7$	2	48	OK	137ms	26MB
$V_8 \circ V_8$	2	35	OK	106ms	27MB
$V_8 \circ V_9$	2	45	OK	133ms	27MB
$V_9 \circ V_{10a}$ (corrected)	2	71	OK	127ms	27MB
$V_9 \circ V_{10a-old}$ (old)	2	71	failed	146ms	27MB
$V_{10a} \circ V_{10b}$ (corrected)	2	120	OK	114ms	27MB
$V_{10b} \circ V_{10c}$ (corrected)	2	144	OK	129ms	27MB
$V_{10b} \circ V_{10c-old}$ (old)	2	144	failed	154ms	26MB

our implementation, in which a substantial portion of the process memory is occupied by the C++ runtime and supporting libraries rather than benchmark-dependent data structures.

Overall, the evaluation demonstrates that the proposed framework can automatically verify the functional correctness of RUS protocols from the literature, detect erroneous implementations, and scale to composed protocols.

8 Discussion and Related Works

8.1 Discussion

This paper introduced an automata-theoretic framework for verifying the functional correctness of repeat-until-success (RUS) protocols. Our approach represents sets of quantum states using level-synchronized tree automata and reduces the verification problem to a small number of choice-aware language inclusion checks. Our experiments demonstrate that the framework can verify RUS protocols from the literature, detect subtle implementation errors, and scale to compositions of multiple RUS circuits with negligible computational cost. To our knowledge, this is the first fully automatic method capable of certifying the functional correctness of RUS protocols.

Beyond the specific case of RUS protocols, the techniques developed in this work have broader applications. The automata-based representation of quantum states and the choice-aware inclusion relation may be used to verify other classes of quantum programs involving measurements and classical control. For instance, the framework could be applied to reasoning about equivalence between quantum circuits or programs, verifying the functional correctness of measurement-based protocols, and analyzing adaptive quantum algorithms.

8.2 Related Works

We compare only against approaches that could in principle handle quantum programs with control flow.

Automata-based Verification. Our work builds on and significantly extends prior automata-theoretic approaches to quantum verification [4, 10], in particular AutoQ [9, 11]. While AutoQ demonstrates that automata-theoretic techniques can be effective for reasoning about quantum circuits, its specification mechanism remains essentially extensional. In particular, correctness is checked by testing the circuit on a finite set of representative inputs (e.g., basis states), without enforcing a one-to-one correspondence between inputs and outputs. As a consequence, such approaches may accept implementations that produce the correct set of outputs but mismatch individual inputs and outputs.

Our framework addresses this limitation by equipping LSTAs with a *choice-sequence semantics*. This semantics records, for each accepting run, a sequence of choice sets that synchronizes the computation across levels of the tree. Based on this, we define *choice-aware inclusion*, which strengthens language inclusion by requiring the compared runs to have compatible choice sequences, thereby enforcing input–output correspondence.

This distinction is crucial in practice. In our experiments, we identify subtle bugs in previously published RUS constructions that were not detected by the specifications verified in [11].

Recent work on *synchronized weighted tree automata* (SWTA) and parameterized verification [2] can be viewed as a weighted extension of level-synchronized tree automata. In SWTA, transitions are labeled with *colors* rather than *sets of choices*, and accepting runs induce a *color sequence* associated with each tree (function). This plays a role analogous to the choice sequences in our framework, in that it records how computations are synchronized across different parts of the tree.

However, there are several important differences between the two approaches. First, quantum operations in the SWTA framework are defined via weighted tree transducers (WTT), and their application to automata is constructed using a product construction of automata. Since WTTs do not carry color information, these transformations are trivially color-sequence preserving. Second, the verification problems considered in the two frameworks differ substantially. In the SWTA setting, color-aware equivalence between SWTAs is reduced to the zero-invariant problem of a linear transition system and solved using Karr's algorithm. By contrast, we design an on-the-fly, simulation-based algorithm for checking it, which is tailored to our semantics and supports efficient verification in practice. It is also worth noting that language inclusion for SWTAs in the usual sense is undecidable, whereas for LSTAs is decidable [4]. Third, the modeling capabilities of the two frameworks target different classes of systems. SWTA is designed to represent families of quantum states parameterized by the size of the system, and is primarily used to reason about circuit families. At present, it does not support measurement nor classical control, and therefore cannot express programs with probabilistic branching nor unbounded looping.

Symbolic Approaches. Another line of work includes tools such as Qbricks [8], which reason about quantum programs using symbolic representations of amplitudes, for instance via path-sum formulations [5]. In these approaches, a program is translated into algebraic expressions, and correctness is reduced to solving symbolic constraints using automated theorem provers or SMT solvers. Qbricks is capable of handling programs with classical control, including conditional branching and bounded iteration. This enables reasoning about programs beyond fixed circuits.

However, reasoning about unbounded loops like RUS protocols is not handled automatically within this framework and typically requires additional invariants or manual guidance.

Deductive Quantum Circuit and Program Verifiers. A substantial body of work on quantum program verification is based on *quantum Hoare logic* [14, 21–23]. In these approaches, specifications are expressed as predicates over quantum states, typically represented as subspaces or density operators, and correctness is established via Hoare triples of the form $\{P\}\, C\, \{Q\}$. Such methods are highly expressive and can reason about general quantum programs, including those with measurements and classical control.

However, these techniques are inherently deductive. Verification typically requires user-supplied invariants and intermediate assertions, and proofs are often carried out interactively or with limited automation. The notion of correctness captured by quantum Hoare logic differs from the *functional correctness* considered in this paper. Hoare triples in these systems are *extensional*: they assert that every output state produced from a precondition lies within a specified set. They do not, in general, enforce a correspondence between individual input and output states. In contrast, our functional triples express a *relational* property, requiring that each input state is mapped to its corresponding output state. This correspondence is tracked explicitly via choice sequences in level-synchronized tree automata and enforced through choice-aware inclusion. Such a notion is essential for verifying RUS protocols, where correctness depends on preserving the intended transformation for every input.

Summary. In contrast to existing approaches, our framework provides a fully automated method for verifying RUS programs. It supports relational specifications of functional correctness, synthesizes loop invariants automatically (in the case of RUS protocols), and reduces verification to decidable automata-theoretic inclusion checks. To the best of our knowledge, this is the first approach capable of automatically verifying RUS protocols in this setting and of uncovering previously unreported errors in the literature.

Statement on the Use of AI Tools

AI tools were used solely as a sounding board and for limited administrative assistance, such as grammar checking. All scientific content, research ideas, technical descriptions, and final text were developed, written, and approved by the authors.

Data-Availability Statement

An environment with the tools and data used for the experimental evaluation in the current study is available at [16].

Acknowledgements

We thank the anonymous reviewers for their feedback that helped to improve the quality of the paper. This work was supported by the Czech Science Foundation project 25-18318S, the FIT BUT internal project FIT-S-26-9011, National Science and Technology Council, R.O.C., projects NSTC 114-2221-E-027-004-MY2, NSTC 115-2221-E-001-003-MY3, NSTC 115-2119-M-001-002, Foxconn Research Award 05T-1120327-2C, and Academia Sinica Investigator Award AS-IV-114-M07-ASSA.

References

[1] Parosh Aziz Abdulla, Yo-Ga Chen, Yu-Fang Chen, Lukáš Holík, Ondřej Lengál, Jyun-Ao Lin, Fang-Yi Lo, and Wei-Lun Tsai. [n. d.]. Verifying Quantum Circuits with Level-Synchronized Tree Automata (Technical Report). https://arxiv.org/abs/2410.18540. arXiv:2410.18540 [cs.LO] doi:10.48550/arXiv.2410.18540

[2] Parosh Aziz Abdulla, Yu-Fang Chen, Michal Hečko, Lukáš Holík, Ondřej Lengál, Jyun-Ao Lin, and Ramanathan S. Thinniyam. 2026. Parameterized Verification of Quantum Circuits. *Proc. ACM Program. Lang.* 10, POPL, Article 70 (Jan. 2026), 30 pages. doi:10.1145/3776712

[3] Parosh Aziz Abdulla, Yo-Ga Chen, Yu-Fang Chen, Lukáš Holík, Ondřej Lengál, Jyun-Ao Lin, Fang-Yi Lo, and Wei-Lun Tsai. 2025. Verifying Quantum Circuits with Level-Synchronized Tree Automata. *Proceedings of the ACM on Programming Languages* 9, POPL (2025), 923–953. doi:10.1145/3704868

[4] Parosh Aziz Abdulla, Yo-Ga Chen, Yu-Fang Chen, Lukáš Holík, Ondřej Lengál, Jyun-Ao Lin, Fang-Yi Lo, and Wei-Lun Tsai. 2025. Verifying Quantum Circuits with Level-Synchronized Tree Automata. *Proc. ACM Program. Lang.* 9, POPL, Article 32 (Jan. 2025), 31 pages. doi:10.1145/3704868

[5] Matthew Amy. 2018. Towards Large-scale Functional Verification of Universal Quantum Circuits. In *Proceedings 15th International Conference on Quantum Physics and Logic, QPL 2018, Halifax, Canada, 3-7th June 2018 (EPTCS, Vol. 287)*, Peter Selinger and Giulio Chiribella (Eds.). 1–21. doi:10.4204/EPTCS.287.1

[6] Pablo Andrés-Martínez and Chris Heunen. 2022. Weakly measured while loops: peeking at quantum states. *Quantum Science and Technology* 7, 2 (feb 2022), 025007. doi:10.1088/2058-9565/ac47f1

[7] Alex Bocharov, Martin Roetteler, and Krysta M. Svore. 2015. Efficient Synthesis of Universal Repeat-Until-Success Quantum Circuits. *Phys. Rev. Lett.* 114 (Feb 2015), 080502. Issue 8. doi:10.1103/PhysRevLett.114.080502

[8] Christophe Chareton, Sébastien Bardin, François Bobot, Valentin Perrelle, and Benoît Valiron. 2021. An Automated Deductive Verification Framework for Circuit-building Quantum Programs. In *Programming Languages and Systems*, Nobuko Yoshida (Ed.). Springer International Publishing, Cham, 148–177. doi:doi.org/10.1007/978-3-030-72019-3_6

[9] Yu-Fang Chen, Kai-Min Chung, Ondrej Lengál, Jyun-Ao Lin, and Wei-Lun Tsai. 2023. AutoQ: An Automata-Based Quantum Circuit Verifier. In *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 13966)*, Constantin Enea and Akash Lal (Eds.). Springer, 139–153. doi:10.1007/978-3-031-37709-9_7

[10] Yu-Fang Chen, Kai-Min Chung, Ondrej Lengál, Jyun-Ao Lin, Wei-Lun Tsai, and Di-De Yen. 2023. An Automata-Based Framework for Verification and Bug Hunting in Quantum Circuits. *Proc. ACM Program. Lang.* 7, PLDI (2023), 1218–1243. doi:10.1145/3591270

[11] Yu-Fang Chen, Kai-Min Chung, Min-Hsiu Hsieh, Wei-Jia Huang, Ondřej Lengál, Jyun-Ao Lin, and Wei-Lun Tsai. 2025. AutoQ 2.0: From Verification of Quantum Circuits to Verification of Quantum Programs. In *Tools and Algorithms for the Construction and Analysis of Systems*, Arie Gurfinkel and Marijn Heule (Eds.). Springer Nature Switzerland, Cham, 87–108. doi:10.1007/978-3-031-90660-2_5

[12] Yu-Fang Chen, Kai-Min Chung, Ondřej Lengál, Jyun-Ao Lin, Wei-Lun Tsai, and Di-De Yen. 2025. An Automata-Based Framework for Verification and Bug Hunting in Quantum Circuits. *Commun. ACM* 68, 6 (June 2025), 85–93. doi:10.1145/3725728

[13] Andrew Cross, Ali Javadi-Abhari, Thomas Alexander, Niel De Beaudrap, Lev S Bishop, Steven Heidel, Colm A Ryan, Prasahnt Sivarajah, John Smolin, Jay M Gambetta, et al. 2022. OpenQASM 3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing* 3, 3 (2022), 1–50. doi:10.1145/3505636

[14] Yuan Feng and Mingsheng Ying. 2021. Quantum Hoare logic with classical variables. *ACM Transactions on Quantum Computing* 2, 4 (2021), 1–43. doi:10.1145/3456877

[15] Mark Koch, Alan Lawrence, Kartik Singhal, Seyon Sivarajah, and Ross Duncan. 2025. GUPPY: pythonic quantum-classical programming. *arXiv preprint arXiv:2510.12582* (2025). doi:10.48550/arXiv.2510.12582

[16] Jyun-Ao Lin, Yu-Fang Chen, Jakub Havlík, Ondřej Lengál, Fang-Yi Lo, Wei-Lun Tsai, and You-Jie Wu. 2026. Verifying repeat-until-success protocols within automata (artifact). doi:10.5281/zenodo.21428108

[17] MS Moreira, Gian Giacomo Guerreschi, Wouter Vlothuizen, Jorge F Marques, Jeroen van Straten, Shavindra P Premaratne, Xiang Zou, Hany Ali, Nandini Muthusubramanian, Christos Zachariadis, et al. 2023. Realization of a quantum neural network using repeat-until-success circuits in a superconducting quantum processor. *npj Quantum Information* 9, 1 (2023), 118. doi:10.1038/s41534-023-00779-5

[18] Michael A. Nielsen and Isaac L. Chuang. 2011. *Quantum Computation and Quantum Information: 10th Anniversary Edition* (10th ed.). Cambridge University Press, USA. https://dl.acm.org/doi/10.5555/1972505

[19] Adam Paetznick and Krysta M. Svore. 2014. Repeat-until-Success: Non-Deterministic Decomposition of Single-Qubit Unitaries. *Quantum Info. Comput.* 14, 15–16 (nov 2014), 1277–1301. https://dl.acm.org/doi/abs/10.5555/2685179.2685181

[20] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. 2018. Q# enabling scalable quantum computing and development with a high-level dsl. In *Proceedings of the real world domain specific languages workshop 2018*. 1–10. doi:10.1145/3183895.3183901

[21] Dominique Unruh. 2019. Quantum Hoare logic with ghost variables. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 1–13. doi:10.1109/LICS.2019.8785779

[22] Mingsheng Ying. 2012. Floyd-Hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 33, 6 (2012), 1–49. doi:10.1145/2049706.2049708

[23] Li Zhou, Nengkun Yu, and Mingsheng Ying. 2019. An applied quantum Hoare logic. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 1149–1162. doi:10.1145/3314221.3314584

Received 2026-03-17; accepted 2026-08-06