

# Parameterized Verification of Quantum Circuits

PAROSH AZIZ ABDULLA, Uppsala University, Sweden and Mälardalen University, Sweden

YU-FANG CHEN, Academia Sinica, Taiwan

MICHAL HEČKO, Brno University of Technology, Czech Republic

LUKÁŠ HOLÍK, Brno University of Technology, Czech Republic and Aalborg University, Denmark

ONDŘEJ LENGÁL, Brno University of Technology, Czech Republic

JYUN-AO LIN, National Taipei University of Technology, Taiwan

RAMANATHAN S. THINNIYAM, Uppsala University, Sweden

We present the first fully automatic framework for verifying relational properties of *parameterized quantum programs*, i.e., a program that, given an input size, generates a corresponding quantum circuit. We focus on verifying input-output correctness as well as equivalence. At the core of our approach is a new automata model, synchronized weighted tree automata (SWTAs), which compactly and precisely captures the infinite families of quantum states produced by parameterized programs. We introduce a class of transducers to model quantum gate semantics and develop composition algorithms for constructing transducers of parameterized circuits. Verification is reduced to functional inclusion or equivalence checking between SWTAs, for which we provide decision procedures. Our implementation demonstrates both the expressiveness and practical efficiency of the framework by verifying a diverse set of representative parameterized quantum programs with verification times ranging from milliseconds to seconds.

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:

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 (January 2026), 30 pages. <https://doi.org/10.1145/3776712>

## 1 Introduction

Quantum technology is advancing at an unprecedented pace and has the potential to reshape numerous sectors at both national and global levels. As quantum computers become increasingly complex and widely deployed, ensuring the correctness of programs that run on them becomes a matter of critical importance. Errors in quantum programs can have serious consequences, including incorrect outcomes in cryptographic tasks, unnecessary consumption of quantum resources, and the misinterpretation of experimental data. Over the past decade, quantum program verification has emerged as a vibrant research area and has seen impressive advances. Notable developments

Authors' Contact Information: [Parosh Aziz Abdulla](mailto:parosh@it.uu.se), Uppsala University, Uppsala, Sweden and Mälardalen University, Västerås, Sweden, [parosh@it.uu.se](mailto:parosh@it.uu.se); [Yu-Fang Chen](mailto:yfc@iis.sinica.edu.tw), Academia Sinica, Taipei, Taiwan, [yfc@iis.sinica.edu.tw](mailto:yfc@iis.sinica.edu.tw); [Michal Hečko](mailto:ihecko@fit.vut.cz), Brno University of Technology, Brno, Czech Republic, [ihecko@fit.vut.cz](mailto:ihecko@fit.vut.cz); [Lukáš Holík](mailto:holik@fit.vutbr.cz), Brno University of Technology, Brno, Czech Republic and Aalborg University, Aalborg, Denmark, [holik@fit.vutbr.cz](mailto:holik@fit.vutbr.cz); [Ondřej Lengál](mailto:lengal@fit.vutbr.cz), Brno University of Technology, Brno, Czech Republic, [lengal@fit.vutbr.cz](mailto:lengal@fit.vutbr.cz); [Jyun-Ao Lin](mailto:jalin@ntut.edu.tw), National Taipei University of Technology, Taipei, Taiwan, [jalin@ntut.edu.tw](mailto:jalin@ntut.edu.tw); [Ramanathan S. Thinniyam](mailto:ramanathan.s.thinniyam@it.uu.se), Uppsala University, Uppsala, Sweden, [ramanathan.s.thinniyam@it.uu.se](mailto:ramanathan.s.thinniyam@it.uu.se).



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2475-1421/2026/1-ART70

<https://doi.org/10.1145/3776712>

include rigorous proof systems, automated reasoning tools, and foundational theories addressing various quantum paradigms; see, e.g., [4, 7, 8, 15, 16, 21, 31, 32, 41, 49, 55, 59–61]. These works have collectively laid the groundwork for understanding and verifying quantum programs in practical and theoretical settings.

Most useful quantum algorithms (e.g., Shor’s, Grover’s, QFT) are inherently *parameterized*<sup>1</sup>, meaning they are designed to operate correctly for arbitrary input sizes. Writing a separate quantum program for each input size of an algorithm is both inefficient and fundamentally unscalable. A more effective approach is to design a *parameterized program* that takes the input size  $n$  and dynamically generates the corresponding quantum circuit.

Yet, existing verification approaches are very limited in their ability to automatically **verify parameterized programs**. Even though notable progress in this direction has been made by the QBRICKS [15] and AUTOQ [4, 21] projects, QBRICKS is not fully automated, and AUTOQ is limited to a narrow class of parameterized circuits. As of now, a fully automatic verification approach capable of handling general parameterized quantum programs remains an open problem. Achieving this goal requires overcoming fundamental challenges, and we address those in this paper.

We particularly focus on automated verification of *relational properties*, aka *functional verification*, that is particularly meaningful and broadly applicable in the context of parameterized quantum programs. Given a set of quantum state pairs  $(p, q)$ , the task is to determine whether the program, for each input size, correctly maps input state  $p$  to the expected output state  $q$ . This captures *input-output correctness* across all parameterized instances.

The main challenges can be narrowed down to delivering three key components of the functional verification framework: (i) a formal model for specifying the property of interest—namely, an infinite set of quantum state pairs; (ii) a language for describing the parameterized quantum program; and (iii) an efficient algorithm for checking whether the program satisfies the given property. We answer these challenges in this paper and present, for the first time, a complete and efficient verification framework tailored for parameterized functional verification.

**First, at the core of our approach is a novel class of automata, which we refer to as *synchronized weighted tree automata (SWTAs)*.** This model extends standard tree automata [27] with two key features: *colors*, for synchronization (inspired by [5]), and *weights*, to encode quantum *amplitudes*<sup>2</sup>. This combination yields a powerful and compact formalism that can represent rich families of quantum state spaces—including those parameterized by input size and those with exponentially many distinct amplitudes. Notably, unlike standard tree automata, which merely characterize a set of quantum states (trees), SWTAs also define a function from *color sequences* to quantum states. This functional view is central to enabling relational verification. Since SWTAs are a new formal model, we also explore its basic properties, such as decidability and complexity of emptiness checking (PSPACE-complete; Section 5.2), inclusion and equivalence checking (both undecidable; Section 5.5), and closure properties (closed under union, not closed under intersection and complement; Section 5.6).

**Second, to reason about circuit behaviors, we introduce a class of transducers that operate on SWTAs.** These transducers serve as semantic models capable of describing the behavior of individual quantum gates, circuits composed of finitely many components, and—crucially—*parameterized programs*. Our transducers provide a uniform representation of standard quantum

<sup>1</sup>In quantum computing, “parameterized circuits/programs” often refers to circuits with parameters such as rotation angles. Here, we use the term to mean circuits parameterized by input size, a usage more familiar in the verification community.

<sup>2</sup>A quantum amplitude is a complex number that influences the chance of seeing a particular result when we observe the quantum state.

gates<sup>3</sup>, including all single-qubit gates, controlled gates, and their compositions. Notably, the encoding is efficient. As a case in point, the Quantum Fourier Transform (QFT) gate can be captured using a transducer with a polynomial number of states, thereby avoiding the anticipated exponential blow-up. Importantly, applying a transducer that encodes a parameterized quantum program to an SWTA preserves the color sequence associated with each state. This preserved sequence acts as a bridge, linking states in two SWTAs that represent the quantum system before and after executing the parameterized program, hence allowing the capture of input-output relations.

We also develop a comprehensive toolkit to support the construction of the transducer of quantum programs. Central to this is a *composition operator* that enables the automatic assembly of complex transducers from simpler components. Given a quantum program and transducers specifying the behavior of individual gates, the composition operator automatically constructs a transducer representing the behavior of the entire program. More generally, it can compose the transducers for two sub-circuits to produce one for the entire circuit. We further extend this operator to parameterized programs: starting from a transducer for a finite circuit, we automatically generate a transducer for a parameterized version consisting of an arbitrary number of repeated components.

**Third, we develop decision procedures for checking equivalence and entailment of SWTAs.** That is, given SWTAs  $\mathcal{A}$  and  $\mathcal{B}$ , we can check that they define the same function, denoted  $\llbracket \mathcal{A} \rrbracket = \llbracket \mathcal{B} \rrbracket$ , or that one is included in the other, denoted  $\llbracket \mathcal{A} \rrbracket \subseteq \llbracket \mathcal{B} \rrbracket$ .

These algorithms complete the foundation of a framework for solving the relational verification problem. Specifically:

- (1) The desired relational property is specified using a pair of SWTAs,  $\mathcal{A}$  and  $\mathcal{B}$ , with related quantum states linked via shared color sequences;
- (2) The parameterized quantum program is encoded as a transducer and applied to  $\mathcal{A}$  to produce a new SWTA,  $\mathcal{A}'$ ;
- (3) Verification is then reduced to checking the inclusion  $\llbracket \mathcal{A}' \rrbracket \subseteq \llbracket \mathcal{B} \rrbracket$ .

Our approach also supports **checking the equivalence of two parameterized quantum programs**—that is, verifying whether they produce identical outputs on all valid inputs, regardless of the input size. This capability is particularly important for ensuring the correctness of program optimizations. The idea behind our method is straightforward:

- (1) We begin by constructing an SWTA  $\mathcal{A}$  that encodes all *computational basis states*, or more generally, any set of  $2^n$  linearly independent quantum states, for every input size  $n$ .
- (2) We then apply the transducers corresponding to the two parameterized programs to  $\mathcal{A}$ , producing two output SWTAs,  $\mathcal{B}$  and  $\mathcal{B}'$ .
- (3) Finally, we check whether the two resulting state sets are equal, i.e., whether  $\llbracket \mathcal{B} \rrbracket = \llbracket \mathcal{B}' \rrbracket$ .

This procedure is sound because quantum programs (or circuits) implement linear transformations. If two such programs produce the same outputs when applied to a complete set of linearly independent inputs, then they are equivalent.

We implemented our framework as a C++ tool named AUTOQ-PARA [3] and evaluated it on a diverse set of parameterized quantum verification tasks. For functional verification, we applied AUTOQ-PARA to the Bernstein-Vazirani algorithm, an arithmetic ripple-carry adder, and a syndrome extraction circuit from a quantum error-correcting code. For equivalence checking, we used AUTOQ-PARA to verify the correctness of optimized implementations of Grover's search algorithm and Hamiltonian simulation by comparing them against their unoptimized, parameterized counterparts.

These benchmarks span a broad spectrum of parameterized quantum circuits, including algorithmic, arithmetic, simulation, and error-correction components, demonstrating the generality and

<sup>3</sup>Quantum gates are the fundamental operations in quantum programs, responsible for transforming quantum states from one form to another.

expressiveness of our approach. AUTOQ-PARA successfully verified all examples efficiently, with runtimes ranging from 14 ms (Bernstein-Vazirani) to 11 s (Adder). Overall, these results showcase the practical effectiveness, versatility, and scalability of our verification framework. To the best of our knowledge, we are the first to fully automatically verify parameterized versions of the circuits.

## 2 Quantum States and Tree Automata

We present a step-by-step overview of quantum states and our automata model. We will illustrate the concepts through a series of simple examples. We begin by introducing a tree-like representation of quantum states, followed by a tree automata-based framework for characterizing (potentially infinite) sets of such states. Next, we introduce a formalism based on (*tree*) *transducers* to capture the transition relation induced by a quantum circuit on sets of quantum states.

### 2.1 Quantum States

Classical and quantum systems differ fundamentally in how they represent and evolve states. A classical state of a system with  $n$  bits is a single element from the set  $\{0, 1\}^n$ . At any given time, the system is in exactly one state (e.g., 0101 or 1110). Quantum computers use qubits (quantum bits) to store and process information. In the quantum setting, classical states are referred to as (*computational*) *basis states*. An  $n$ -qubit system is described by a quantum state belonging to a  $2^n$ -dimensional space, and a state is a *superposition* of all  $2^n$  basis states. Each basis state, usually written as in the *Dirac notation*  $|x\rangle$ , where  $x \in \{0, 1\}^n$ , corresponds to a definite state of all  $n$  qubits (with each qubit being either  $|0\rangle$  or  $|1\rangle$ ). A general state is written as  $|\psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$ , where each  $x$  is a basis state, and the coefficients  $\alpha_x$ , called the *amplitudes*, are complex numbers satisfying the normalization condition:  $\sum_{x \in \{0,1\}^n} |\alpha_x|^2 = 1$ . Fig. 1 depicts a quantum state in a system with 3 qubits. In the quantum computing literature, states are sometimes written as column vectors of length  $2^n$ , with complex entries corresponding to the amplitudes of the respective basis states. To simplify notation, we often use the transpose of this column vector. For example, we write  $\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}^T$  to represent the state  $\frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle$ . This results in a more compact row vector form. In our verification framework, we will represent the state of  $n$  qubits as a perfect binary tree of depth  $n$ , where each leaf represents the amplitude of the basis state we get by traversing the tree from the root to the leaves (Fig. 1). The tree representation is closely related to the vector representation of quantum states. A tree encodes the vector by concatenating the amplitudes of the leaves in which they occur from left to right.

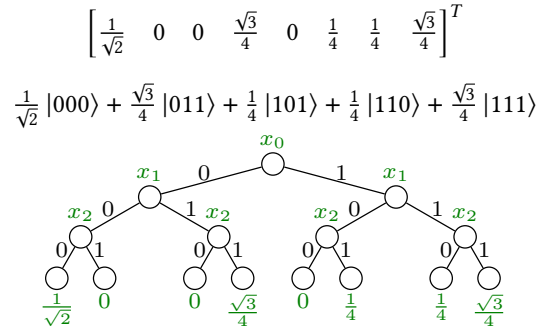


Fig. 1. A quantum state involving three qubits. We provide the vector, Dirac, and tree-based representations. The tree spans all basis states, with the corresponding amplitudes shown as leaf labels. For instance, the left-most path represents  $|000\rangle$  with amplitude  $\frac{1}{\sqrt{2}}$ .

### 2.2 Tree Automata

We use (tree) automata to encode both finite and infinite sets of quantum states. We will introduce our tree automata model in three steps, namely, plain (standard) automata [21], level-synchronized (colored) tree automata (LSTAs) [4], and weighted synchronized tree automata (SWTAs). A plain automaton adheres to the classical definition, comprising a finite set of states and transition rules.

Notice that the classical definition attaches a label to each transition. To simplify the presentation of the overview section, we retain only the leaf transition labels and omit the non-leaf labels. We use the leaf transition labels to represent amplitudes. The non-leaf labels are assumed to be identical and irrelevant at this stage. These non-leaf symbols will later be used to represent variable names and, in some cases, to avoid non-deterministic transitions. As usual, the automaton accepts a given tree if it is possible to label the nodes of the tree with states of the automaton such that the root is labeled by the root state and the children of each node are labeled in accordance with the transition rules. Fig. 2 depicts an automaton accepting trees that characterize all basis quantum states over three qubits. Intuitively, the states labeled  $q_0^i$  accept trees whose leaves all have amplitude 0, while the states labeled  $q_1^i$  accept trees with exactly one leaf having amplitude 1, and all other leaves having amplitude 0. Analogously, we can construct automata that handle any given number  $n$  of qubits. A key observation is that plain tree automata allow us to represent a set containing exponentially many quantum states using only a linear number of automaton states. On the other hand, plain automata cannot capture infinite sets of quantum states. The reason for this is that we represent states as *perfect* binary trees, and representing an infinite set of such trees requires a synchronization mechanism: all leaf transitions must occur at the same level.

To handle such limitations, [4] introduces the LSTA model in which each transition is annotated with a color. The set of colors is used to enforce level-synchronization: all transitions applied at a given level of the accepted tree must be labeled with the same color. LSTAs can represent sets of trees more compactly than plain tree automata. Let us consider the example shown in Fig. 3. The automaton accepts the two trees depicted in Fig. 3c as follows. It begins with a single transition from the root state  $q_0$ ; the color of this transition is immaterial. The transition  $q_0 \rightarrow (q_1, q_1)$  indicates that, at the next level, the state  $q_1$  is used to generate both the left and the right subtree. Note that  $q_1$  has two transitions, one colored red (and decorated by ①) and the other blue (decorated by ②). However, due to the color restriction, we must apply either the ①-transition to both children or the ②-transition to both. We cannot mix the colors, since all transitions applied at a given level must share color (red or blue, in this case).

The next example, depicted in Fig. 4, demonstrates the expressive power of LSTAs compared to plain tree automata. The LSTA accepts the set of all basis states  $\{|x\rangle \mid x \in \{0, 1\}^*\}$  for an arbitrary number  $n$  of qubits. Recall that the plain automaton in Fig. 2 characterizes the set of all basis states for the fixed case  $n = 3$ . To represent the basis states for arbitrary  $n$ , we would

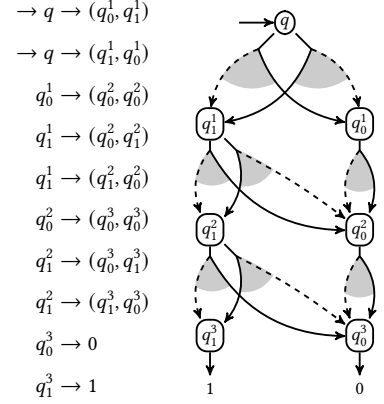


Fig. 2. A plain tree automaton that characterizes the set of all 3-qubit basis states. In the figure, transitions are represented by hyperedges (represented by grey arcs) with dashed arrows going to the left-hand child and solid arrows going to the right-hand child.

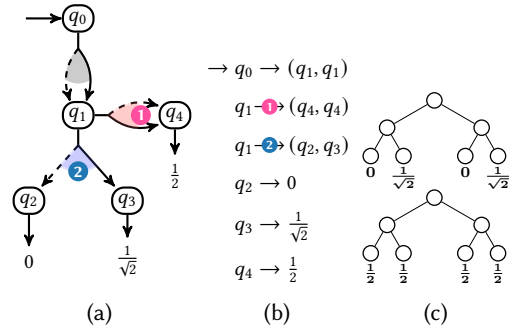


Fig. 3. An LSTA. (a) A picture of the automaton. (b) The set of transitions. (c) The two trees accepted by the automaton.

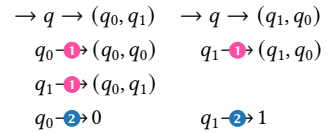


Fig. 4. An LSTA recognizing the set of all basis states.

need infinitely many plain automata—one for each value of  $n$ . In contrast, a single LSTA suffices to describe the entire set. We use the state  $q_0$  to generate trees in which all leaves have amplitude 0, and the state  $q_1$  to generate trees with exactly one leaf of amplitude 1, while all other leaves have amplitude 0. The states  $q_0$  and  $q_1$  allow transitions that are either red or blue. Intuitively, red transitions are applied to inner nodes, while blue transitions are applied at the leaf level. Since all transitions applied at the same level must have the same color, we ensure that only perfect binary trees are generated: we apply red transitions repeatedly, level by level, to build the tree, and at some point, switch to blue transitions to generate the leaves.

Finally, we introduce *synchronized weighted tree automata*, and demonstrate their superiority over the LSTA model. We begin by considering the set of all quantum basis states in *uniform superposition*. These are the states where all leaves have identical amplitudes. Figs. 5b and 5c depict the trees corresponding to the cases where the number of qubits is 1 and 2, respectively. For  $n$  qubits, the amplitudes are equal to  $2^{-\frac{n}{2}}$ , i.e., they depend on the number of qubits. This uniform superposition set cannot

be captured by any finite set of LSTAs, since we would need one set of transitions to express the amplitudes for each given value of  $n$ . To overcome this limitation, we introduce *synchronized weighted tree automata* (SWTAs), and show how weights can be used—together with colors—to generate all quantum states in uniform superposition. Recall that in both plain and colored automata, a transition of the form  $q \rightarrow (q_1, q_2)$  defines the set of trees where the left and right subtrees are generated from the states  $q_1$  and  $q_2$ , respectively. In contrast, SWTA allows for a more general form of transition, where the subtrees are generated using *linear combinations* of states. These linear combinations are constructed by *adding* states and *multiplying* them by complex number *constants*. The SWTA depicted in Fig. 5 provides a simple example. To construct the left and right subtrees, we (i) recursively generate trees from the state  $q$ , and (ii) multiply the resulting trees by the scalar  $\frac{1}{\sqrt{2}}$ .

This scaling means that each leaf of the subtree is multiplied by the constant  $\frac{1}{\sqrt{2}}$ . As before, the construction can be repeated an arbitrary number of times using red transitions, after which we apply the blue transition to generate the leaf nodes labeled  $\ell$ . A leaf node has the weight 1 by default. This is in contrast to the standard notion of tree automata, where leaf rules explicitly assign values to leaves. In our setting, the use of weights makes explicit leaf values unnecessary. Notice that weights and colors operate *in tandem* to generate the desired set of trees. Without colors, we cannot enforce the level-wise synchronization required by the uniform superposition; and without weights, we cannot generate the correct leaf amplitudes.

### 2.3 Transducers

We model the behavior of a quantum circuit as a transition relation over a set of trees. This transition relation is encoded as a transducer that transforms a tree representing the input quantum state into a tree representing the result of applying the circuit to that input. Since a tree encodes a quantum state by storing the sequence of amplitudes at its leaves, the transducer specifies how these amplitudes are modified to produce the output tree. A transducer consists of a finite set of states and a collection of transition rules. To initiate the transformation, the root of the input tree is labeled with a root state. Each state, assigned to a node in the input tree, prescribes how to construct the left and right subtrees of the output tree based on the structure of the corresponding

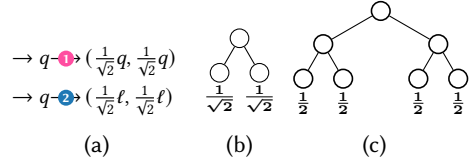


Fig. 5. (a) An SWTA recognizing the set of all quantum states with uniform superposition;  $\ell$  is a leaf state. (b) and (c) show the trees with one qubit resp. two qubits in uniform superposition.

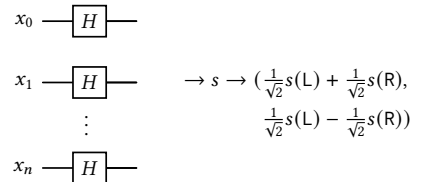
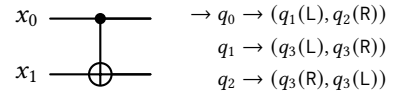
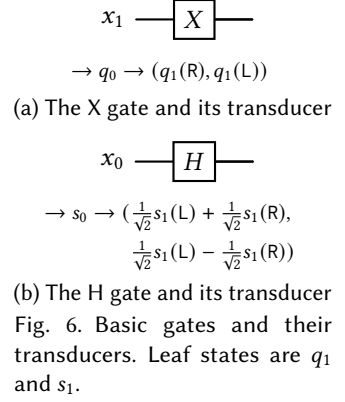
input subtrees. The output tree is constructed by applying linear combinations—defined by the transducer’s transition rules—to the subtrees at each node. The right-hand side of a transition rule is a pair that specifies how to generate the left and right subtrees of the output tree. This is done using a set of *ground terms*, each of the form  $q(L)$  or  $q(R)$ . Intuitively,  $q(L)$  means applying the state  $q$  to the left subtree of the input, thereby producing a corresponding output subtree. Similarly,  $q(R)$  refers to applying  $q$  to the right subtree. A transition defines the new left and right subtrees as linear combinations of such ground terms. We begin by describing transducers that implement individual quantum gates. The X gate (quantum NOT) inverts a single qubit by swapping its amplitudes (Fig. 6a). The corresponding transducer consists of two states,  $q_0$  and  $q_1$ , together with one transition. The transformation begins by labeling the root of the input tree with the root state  $q_0$ . To construct the left subtree of the output, we apply state  $q_1$  to the right subtree of the input; conversely, we apply  $q_1$  to the left subtree to construct the right subtree. The leaf state does not alter the amplitudes—it simply multiplies them by 1. As a result, the transducer swaps the amplitudes of the two leaves, which captures the behavior of the X gate.

The H gate (Hadamard) creates superpositions, enabling quantum algorithms to explore multiple states in parallel. The transducer, depicted in Fig. 6b has two states: the root state  $q_0$  and the leaf state  $q_1$ . To construct the left subtree of the output, we apply  $q_1$  to both the left and right subtrees of the input, multiply each resulting subtree by  $\frac{1}{\sqrt{2}}$ , and then sum them. To construct the right subtree, we again apply  $q_1$  to both subtrees, multiply each by  $\frac{1}{\sqrt{2}}$ , but this time subtract the right subtree from the left. This transformation faithfully models the Hadamard gate, which maps the basis states  $|0\rangle$  and  $|1\rangle$  to superpositions  $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$  and  $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ , respectively.

The CX gate is a two-qubit gate that flips (i.e., applies the NOT operation to) the target qubit if and only if the control qubit is in the quantum basis state  $|1\rangle$ . If the control qubit is in the quantum basis state  $|0\rangle$ , the target qubit remains unchanged. In terms of tree representation, the transducer corresponding to the CX gate preserves the left subtree (representing the control qubit), while it swaps the two subtrees of the right subtree (representing the target qubit). This conditional swap encodes the semantics of the CX gate: the output is modified only when the control is in the quantum basis state  $|1\rangle$ .

## 2.4 Composition

In the paper, we will introduce a set of composition operators on SWTAs and transducers. The first operator applies a transducer  $T$  to an SWTA  $\mathcal{A}_1$  to produce a new SWTA  $\mathcal{A}_2$ . Here,  $\mathcal{A}_1$  describes a *precondition*, i.e., a set of input trees (quantum states), and  $T$  specifies how the circuit transforms these input trees to obtain the *postcondition*, i.e., the resulting set of output trees. The second operator composes a sequence of transducers—each representing an individual gate or a subcircuit—into a single transducer that captures the behavior of the entire circuit. The third operator applies to a given



circuit and captures the *parameterized* circuit obtained by repeating the circuit an unbounded number of times. For instance, given the transducer for a single H gate (as shown in Fig. 6b), this operator produces a parameterized transducer that represents the repeated application of an arbitrary number of H gates (Fig. 8).

Finally, **functional and equivalence verification** can be performed using all of the components introduced above, together with the SWTA functional equivalence and inclusion procedures described in Section 5.3. At this point, we should have a preliminary understanding of the overall framework. In the next step, we will provide more rigorous exposition.

### 3 Formal Definitions

*Basics.* We use  $\mathbb{N}$  to denote  $\{0, 1, \dots\}$ ,  $\mathbb{Z}$  to denote integers, and  $\mathbb{C}$  to denote complex numbers. Given a partial function  $f: A \rightarrow B$ , we use  $f(a) = \perp$  to denote that the value of  $f$  for  $a \in A$  is undefined and  $\text{dom}(f)$  to denote the set  $\{a \in A \mid f(a) \neq \perp\}$ . A *linear form*  $v$  over  $Q = \{q_1, \dots, q_n\}$  is a non-empty partial map from  $Q$  to  $\mathbb{C}$  usually given by a formal sum  $v = \sum_{q_i \in P} a_{q_i} \cdot q_i$  with  $P \subseteq Q$  and each  $a_{q_i} \in \mathbb{C}$  (we often omit the operator ‘ $\cdot$ ’ and the coefficient ‘1’ if present, e.g., we may write “ $3q_1 + q_2$ ”; the linear form “0” represents an empty map). We emphasize that we distinguish the cases when  $q_i \notin P$  and  $q_i \in P$  with  $a_{q_i} = 0$ , e.g., “ $3q_1$ ” and “ $3q_1 + 0q_2$ ” are two different things. We will write  $v[q]$  to denote  $a_q$  (if  $q \notin P$ , then  $v[q] = \perp$ ) and use  $\mathbb{L}_Q$  to denote the set of all linear forms over  $Q$ . The *state support* of  $v$  is defined as  $\text{st}(v) = P$ .

*Words.* An *alphabet* is a finite non-empty set of *symbols*. Given an alphabet  $A$ , a *word* over  $A$  is a finite sequence of symbols  $w = a_1 \dots a_n$  with  $a_i \in A$  for all  $1 \leq i \leq n$ . For two words  $u = a_1 \dots a_n$  and  $v = b_1 \dots b_m$  over  $A$ , we use  $u \bullet v$  (or just the juxtaposition  $uv$ ) to denote the word  $a_1 \dots a_n b_1 \dots b_m$ , with the neutral element  $\epsilon$  (the empty string). Let  $B, C$  be sets of words over  $A$ . Then  $B \bullet C$  (or just the juxtaposition  $BC$ ) is defined as the set  $\{u \bullet v \mid u \in B, v \in C\}$ . We use  $B^0$  to denote the set  $\{\epsilon\}$  and for  $i \in \mathbb{N}$ , we use  $B^{i+1}$  to denote the set  $B \bullet B^i$ . The notation  $B^{<n}$  is used for  $\bigcup_{0 \leq i < n} B^i$  and  $B^*$  denotes  $\bigcup_{0 \leq i} B^i$ . Let us fix a (finite non-empty) alphabet  $\Gamma$  such that  $\Gamma \cap \mathbb{C} = \emptyset$ .

*Trees.* A (*perfect finite*) *binary tree*<sup>4</sup> over  $\Gamma$  is a partial function  $t: \{0, 1\}^* \rightarrow (\Gamma \cup \mathbb{C})$  such that the following conditions hold:

- (1)  $t$  is a finite, non-empty, and prefix-closed partial function and
- (2) there exists a unique  $h \in \mathbb{N}$ , called the *height* of  $t$  and denoted as  $h(t)$ , such that for all positions  $p \in \text{dom}(t)$ :
  - (a) if  $|p| = h$ , then  $t(p) \in \mathbb{C}$  (leaf nodes),
  - (b) if  $|p| < h$ , then  $t(p) \in \Gamma$  (internal nodes), and
  - (c) if  $|p| > h$ , then  $t(p) = \perp$  (undefined).

We use  $\mathbb{T}_\Gamma$  to denote the set of all trees over  $\Gamma$ . For a word  $u \in \{0, 1\}^*$  representing a *branch* of  $t$ , the *subtree* rooted at  $u$  is given as  $t|_u = \{v \mapsto t(uv) \mid uv \in \text{dom}(t)\}$ . For trees  $t_0, t_1 \in \mathbb{T}_\Gamma$  of the same height and a label  $a \in \Gamma$ , we define the tree constructor

$$\text{cons}(a, t_0, t_1) = \{\epsilon \mapsto a\} \cup \{0u \mapsto t_0(u) \mid u \in \text{dom}(t_0)\} \cup \{1u \mapsto t_1(u) \mid u \in \text{dom}(t_1)\}. \quad (1)$$

Trees  $t, t' \in \mathbb{T}_\Gamma$  are *compatible* iff  $h(t) = h(t')$  and for all  $p \in \{0, 1\}^{<h(t)}$  it holds that  $t(p) = t'(p)$ . Given compatible trees  $t, t' \in \mathbb{T}_\Gamma$  and  $a \in \mathbb{C}$ , we define the following operations on trees:

$$t + t' = \{p \mapsto t(p) \mid p \in \{0, 1\}^{<h(t)}\} \cup \{p \mapsto t(p) + t'(p) \mid p \in \{0, 1\}^{h(t)}\} \quad \text{and} \quad (2)$$

$$a \cdot t = \{p \mapsto t(p) \mid p \in \{0, 1\}^{<h(t)}\} \cup \{p \mapsto a \cdot t(p) \mid p \in \{0, 1\}^{h(t)}\}. \quad (3)$$

<sup>4</sup>We emphasize that all trees considered in this paper are *perfect*, i.e., all branches have the same height. This is in contrast to the definition of trees in, e.g., [27]

Intuitively,  $t + t'$  puts the trees over each other and sums up the values in the leaves and  $a \cdot t$  multiplies all values in the leaves of  $t$  by  $a$ . The result of  $t + t'$  for incompatible trees is  $\perp$  and if any of the operands is  $\perp$ , the result is also  $\perp$ .

### 3.1 Synchronized Weighted Tree Automata

A (finite binary) *synchronized weighted tree automaton* (SWTA) over  $\Gamma$  is a tuple  $\mathcal{A} = \langle Q, \delta, \Omega, \text{root}, E \rangle$  where  $Q$  is a finite set of *states*,  $\Omega = \{\mathbf{1}, \dots, \mathbf{n}\}$  is a (finite non-empty) set of *colors*,  $\text{root} \in Q$  is the *root state*,  $E \subseteq Q$  is a set of *leaf states*, and  $\delta: Q \times \Gamma \times \Omega \rightarrow (\mathbb{L}_Q \times \mathbb{L}_Q)$  is a (top-down) *partial transition function*. We also use  $q \xrightarrow{a} a(\ell, \mathbf{r})$  to denote that  $\delta(q, a, \mathbf{c}) = (\ell, \mathbf{r})$ , where  $\ell, \mathbf{r} \in \mathbb{L}_Q$ . An example of an SWTA transition is  $q_1 \xrightarrow{\mathbf{1}} a(\frac{1}{\sqrt{2}}q_1 + \frac{1}{\sqrt{2}}q_2, \frac{1}{\sqrt{2}}q_2 + q_3)$ .

The *tree function of a state*  $q \in Q$  of  $\mathcal{A}$  is a (partial) function  $\llbracket \mathcal{A}, q \rrbracket: (\Gamma \times \Omega)^* \rightarrow \mathbb{T}_\Gamma$  defined inductively as follows:

- (1) (base case)  $\llbracket \mathcal{A}, q \rrbracket(\epsilon)$  is the leaf 1 if  $q$  is a leaf state, otherwise the value is undefined. Formally,

$$\llbracket \mathcal{A}, q \rrbracket(\epsilon) = \begin{cases} \{\epsilon \mapsto 1\} & \text{if } q \in E, \\ \perp & \text{otherwise.} \end{cases} \quad (4)$$

- (2) (inductive case)  $\llbracket \mathcal{A}, q \rrbracket(\langle a, \mathbf{c} \rangle \bullet u) = t$  for  $a \in \Gamma$ ,  $\mathbf{c} \in \Omega$ , and  $u \in (\Gamma \times \Omega)^*$ , where  $t$  is defined in the following way. Let  $q \xrightarrow{a} a(\ell, \mathbf{r}) \in \delta$  with

$$\text{st}(\ell) = \{q_1^\ell, \dots, q_m^\ell\} \quad \text{and} \quad \text{st}(\mathbf{r}) = \{q_1^r, \dots, q_n^r\}. \quad (5)$$

Then  $t = \text{cons}(a, t_\ell, t_r)$  where

$$t_\ell = \sum_{i=1}^m \ell[q_i^\ell] \cdot \llbracket \mathcal{A}, q_i^\ell \rrbracket(u) \quad \text{and} \quad t_r = \sum_{j=1}^n \mathbf{r}[q_j^r] \cdot \llbracket \mathcal{A}, q_j^r \rrbracket(u). \quad (6)$$

Intuitively,  $t$  connects the  $a$ -labeled root to the sub-trees  $t_\ell$  and  $t_r$ , constructed from the linear forms  $\ell$  and  $\mathbf{r}$ , respectively. In each case, every state  $q$  occurring in the linear form is substituted by the tree  $\llbracket \mathcal{A}, q \rrbracket(u)$ , resulting in a linear form over trees, which is then summed up.

The *tree function of  $\mathcal{A}$* , written  $\llbracket \mathcal{A} \rrbracket$ , is a function  $\llbracket \mathcal{A} \rrbracket: (\Gamma \times \Omega)^* \rightarrow \mathbb{T}_\Gamma$  given as  $\llbracket \mathcal{A} \rrbracket = \llbracket \mathcal{A}, \text{root} \rrbracket$ . Note that this definition differs slightly from the simplified version used in the introduction, which describes a tree function as a mapping from color sequences to quantum states. The formal definition maps *symbol-color* sequences to quantum states.

*Example 3.1.* Let  $\mathcal{A}_{\text{ex}}$  be an SWTA defined by the following transitions (with the set of leaf states  $\{u, v\}$ ):

$$\begin{aligned} \rightarrow q \xrightarrow{\mathbf{1}} a(r + s, r - s) & \quad r \xrightarrow{\mathbf{1}} a(2u, 0u) & \quad s \xrightarrow{\mathbf{1}} a(u + v, 0v) \\ \rightarrow q \xrightarrow{\mathbf{2}} a(r - s, r + s) & \quad r \xrightarrow{\mathbf{2}} a(0u, \tfrac{1}{2}u) & \quad s \xrightarrow{\mathbf{2}} a(u - v, u - \tfrac{3}{2}v) \end{aligned} \quad (7)$$

Consider a symbol-color sequence  $w = \langle a, \mathbf{1} \rangle \langle a, \mathbf{2} \rangle$  and let us compute the tree  $t = \llbracket \mathcal{A}_{\text{ex}} \rrbracket(w)$ . For the first symbol-color pair  $\langle a, \mathbf{1} \rangle$  and the root state  $q$ , the relevant transition is  $q \xrightarrow{\mathbf{1}} a(r + s, r - s)$ . Let us therefore compute the sub-trees  $t_r = \llbracket \mathcal{A}_{\text{ex}}, r \rrbracket(\langle a, \mathbf{2} \rangle)$  and  $t_s = \llbracket \mathcal{A}_{\text{ex}}, s \rrbracket(\langle a, \mathbf{2} \rangle)$ . For  $t_r$ , the relevant transition is  $r \xrightarrow{\mathbf{2}} a(0u, \frac{1}{2}u)$ . Since  $u$  is a leaf state, we can conclude that  $t_r = \text{cons}(a, \{\epsilon \mapsto 0\}, \{\epsilon \mapsto \frac{1}{2}\})$ , or  $0 \xrightarrow{a} \frac{1}{2}$ . On the other hand, for  $t_s$ , the relevant transition is  $s \xrightarrow{\mathbf{2}} a(u - v, u - \frac{3}{2}v)$ . Since  $u$  and  $v$  are both leaf states, we have that  $t_s = \text{cons}(a, \{\epsilon \mapsto 1 - 1\}, \{\epsilon \mapsto 1 - \frac{3}{2}\}) = \text{cons}(a, \{\epsilon \mapsto 0\}, \{\epsilon \mapsto -\frac{1}{2}\})$ , or  $0 \xrightarrow{a} -\frac{1}{2}$ . Having computed  $t_r$  and  $t_s$ , we can now construct the final tree  $t$  (using the  $q$ -transition above) as  $\text{cons}(a, t_r + t_s, t_r - t_s) = 0 \xrightarrow{a} \frac{1}{2} \quad 0 \xrightarrow{a} -\frac{1}{2}$ . In a similar manner,

one can easily compute the remaining values of  $\llbracket \mathcal{A}_{ex} \rrbracket$ , summarized below:

$$\begin{aligned} \llbracket \mathcal{A}_{ex} \rrbracket(\langle a, \textcolor{red}{1} \rangle \langle a, \textcolor{red}{1} \rangle) &= 4 \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 0 \end{array} \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 0 \end{array} \text{---} 0 & \llbracket \mathcal{A}_{ex} \rrbracket(\langle a, \textcolor{red}{1} \rangle \langle a, \textcolor{blue}{2} \rangle) &= 0 \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 0 \end{array} \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 1 \end{array} \text{---} 1 \\ \llbracket \mathcal{A}_{ex} \rrbracket(\langle a, \textcolor{blue}{2} \rangle \langle a, \textcolor{red}{1} \rangle) &= 0 \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 4 \end{array} \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 0 \end{array} \text{---} 0 & \llbracket \mathcal{A}_{ex} \rrbracket(\langle a, \textcolor{blue}{2} \rangle \langle a, \textcolor{blue}{2} \rangle) &= 0 \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 1 \quad 0 \end{array} \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 0 \end{array} \text{---} 0 \end{aligned} \quad (8)$$

We say that  $t$  is *accepted from*  $q$  in  $\mathcal{A}$ , denoted as  $t \in \mathcal{L}(\mathcal{A}, q)$ , if there exists a sequence  $w \in (\Gamma \times \Omega)^*$  such that  $\llbracket \mathcal{A}, q \rrbracket(w) = t$ , and say that  $t$  is *accepted by*  $\mathcal{A}$  if  $t \in \mathcal{L}(\mathcal{A}, \text{root})$ . The *language* of  $\mathcal{A}$ , denoted as  $\mathcal{L}(\mathcal{A}) \subseteq \mathbb{T}_\Gamma$ , is the set of all trees accepted by  $\mathcal{A}$ . We say that two SWTAs  $\mathcal{A}$  and  $\mathcal{B}$  are *functionally equivalent* if  $\llbracket \mathcal{A} \rrbracket = \llbracket \mathcal{B} \rrbracket$ , *functionally included* if  $\llbracket \mathcal{A} \rrbracket \subseteq \llbracket \mathcal{B} \rrbracket$  (we see  $\llbracket \mathcal{A} \rrbracket$  and  $\llbracket \mathcal{B} \rrbracket$  as the sets representing the functions), and that they are *language equivalent* if  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ . We note that every two functionally equivalent SWTAs are also language equivalent, but not necessarily the other way round.

*Example 3.2.* Consider the SWTA  $\mathcal{A}_{ex}$  from Example 3.1 and the following SWTA  $\mathcal{B}_{ex}$  with the set of leaf states  $\{h\}$ :

$$\begin{aligned} \rightarrow f \text{---} \textcolor{red}{1} \rightarrow a(4g, 0h) & \quad g \text{---} \textcolor{red}{1} \rightarrow a(0h, \tfrac{1}{4}h) & \quad k \text{---} \textcolor{red}{1} \rightarrow a(4h, 0h) & \quad h \text{---} \textcolor{red}{1} \rightarrow a(h, h) \\ \rightarrow f \text{---} \textcolor{blue}{2} \rightarrow a(0h, k) & \quad g \text{---} \textcolor{blue}{2} \rightarrow a(h, 0h) & \quad k \text{---} \textcolor{blue}{2} \rightarrow a(0h, h) & \quad h \text{---} \textcolor{blue}{2} \rightarrow a(h, h) \end{aligned} \quad (9)$$

The tree function of  $\mathcal{B}_{ex}$  can be computed to be the following:

$$\begin{aligned} \llbracket \mathcal{B}_{ex} \rrbracket(\langle a, \textcolor{red}{1} \rangle \langle a, \textcolor{red}{1} \rangle) &= 0 \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 1 \quad 0 \end{array} \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 0 \end{array} \text{---} 0 & \llbracket \mathcal{B}_{ex} \rrbracket(\langle a, \textcolor{red}{1} \rangle \langle a, \textcolor{blue}{2} \rangle) &= 4 \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 0 \end{array} \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 0 \end{array} \text{---} 0 \\ \llbracket \mathcal{B}_{ex} \rrbracket(\langle a, \textcolor{blue}{2} \rangle \langle a, \textcolor{red}{1} \rangle) &= 0 \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 4 \end{array} \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 0 \end{array} \text{---} 0 & \llbracket \mathcal{B}_{ex} \rrbracket(\langle a, \textcolor{blue}{2} \rangle \langle a, \textcolor{blue}{2} \rangle) &= 0 \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 0 \end{array} \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 1 \end{array} \text{---} 1 \end{aligned} \quad (10)$$

We see that  $\mathcal{L}(\mathcal{A}_{ex}) = \mathcal{L}(\mathcal{B}_{ex}) = \left\{ 0 \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 1 \quad 0 \end{array} \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 0 \end{array} \text{---} 0, 4 \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 0 \end{array} \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 0 \end{array} \text{---} 0, 0 \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 4 \end{array} \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 0 \end{array} \text{---} 0, 0 \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 0 \end{array} \text{---} \textcolor{red}{a} \begin{array}{c} \swarrow \textcolor{red}{a} \searrow \\ 0 \quad 1 \end{array} \text{---} 1 \right\}$ , but  $\llbracket \mathcal{A}_{ex} \rrbracket \neq \llbracket \mathcal{B}_{ex} \rrbracket$ , since, e.g.,  $\llbracket \mathcal{A}_{ex} \rrbracket(\langle a, \textcolor{red}{1} \rangle \langle a, \textcolor{red}{1} \rangle) \neq \llbracket \mathcal{B}_{ex} \rrbracket(\langle a, \textcolor{red}{1} \rangle \langle a, \textcolor{red}{1} \rangle)$ .  $\square$

### 3.2 Weighted Tree Transducers

A (finite binary two-tape) *weighted tree transducer* (WTT) over alphabet  $\Gamma$  is a tuple  $\mathcal{T} = \langle Q, \delta, \text{root}, E \rangle$  where  $Q$  is a finite set of *states*,  $\text{root} \in Q$  is the *root state*,  $E \subseteq Q$  is a set of *leaf states*, and  $\delta: Q \times \Gamma \rightarrow (\mathbb{L}_{Q(\text{L}, \text{R})} \times \mathbb{L}_{Q(\text{L}, \text{R})})$  is a *transition function* where  $Q(\text{L}, \text{R})$  is the set of ground terms of the form  $q(\text{L})$  and  $q(\text{R})$ , for  $q \in Q$  (the symbols L and R used here denote the *Left* and the *Right* subtree of the input tree respectively). An example of a WTT transition is  $q \rightarrow a(\frac{1}{\sqrt{2}}q(\text{L}) + p(\text{R}), -\frac{1}{\sqrt{2}}q(\text{L}) - 3q(\text{R}))$ .

For every state  $q \in Q$ , the transducer defines a (partial) unary function over trees  $\mathcal{T}_q: \mathbb{T}_\Gamma \rightarrow \mathbb{T}_\Gamma$  defined below. First, for a linear form  $x \in \mathbb{L}_{Q(\text{L}, \text{R})}$  and a pair of trees  $t_L$  and  $t_R$ , we use  $x(t_L, t_R)$  to denote the tree obtained by replacing each  $q(\text{L})$  and  $q(\text{R})$  in  $x$  by the tree  $\mathcal{T}_q(t_L)$  and  $\mathcal{T}_q(t_R)$ , respectively, and summing up the resulting linear form over trees (the result can be undefined due to incompatibility). Then  $\mathcal{T}_q$  is defined inductively in the following way:

- (1) (base case) If  $t = \{\epsilon \mapsto a\}$  (i.e.,  $t$  is a leaf) then if  $q \in E$ , then  $\mathcal{T}_q(t) = t$ , else  $\mathcal{T}_q(t) = \perp$  (i.e., it is undefined).
- (2) (inductive case) If  $t \supset \{\epsilon \mapsto a\}$  (i.e.,  $t$  is not a leaf) then if  $\delta(q, a) = \perp$  (i.e., there is no transition for  $q$  and  $a$ ) then  $\mathcal{T}_q(t) = \perp$ , else if  $\delta(q, a) = (\ell, r)$  then  $\mathcal{T}_q(t) = \text{cons}(a, \ell(t_{|0}, t_{|1}), r(t_{|0}, t_{|1}))$  (and it is undefined one of the linear forms evaluates to  $\perp$ ). That is, the two sub-trees of the resulting tree are both transducer images of the sub-trees  $t_{|0}$  and  $t_{|1}$  of the original tree w.r.t. the linear forms.

The transducer  $\mathcal{T}$  then defines the function given as  $\mathcal{T}_{\text{root}}$ .

*Example 3.3.* Let  $\mathcal{T}_{ex}$  be a WTT defined as follows with the set of leaf states  $\{p\}$ :

$$\rightarrow p \rightarrow a(\frac{1}{\sqrt{2}}z(L) + \frac{1}{\sqrt{2}}z(R), \frac{1}{\sqrt{2}}z(L) - \frac{1}{\sqrt{2}}z(R)) \quad z \rightarrow a(p(L), -p(R)) \quad (11)$$

Further, let  $t$  be the tree  $0 \xrightarrow{a} 0 \xrightarrow{a} 1$ . Let us now compute the tree  $t' = \mathcal{T}_{ex}(t)$ . We start with the transition  $p \rightarrow a(\frac{1}{\sqrt{2}}z(L) + \frac{1}{\sqrt{2}}z(R), \frac{1}{\sqrt{2}}z(L) - \frac{1}{\sqrt{2}}z(R))$ . Here,  $t_\ell = 0 \xrightarrow{a} 0$  and  $t_r = 0 \xrightarrow{a} 1$ . Let us now compute the values of  $\mathcal{T}_z(t_\ell)$  and  $\mathcal{T}_z(t_r)$  using the transition  $z \rightarrow a(p(L), -p(R))$ . Since  $p$  is a leaf state, the values are  $t'_\ell = \mathcal{T}_z(t_\ell) = 0 \xrightarrow{a} 0$  and  $t'_r = \mathcal{T}_z(t_r) = 0 \xrightarrow{a} 1$ . The result of the linear form  $\frac{1}{\sqrt{2}}z(L) + \frac{1}{\sqrt{2}}z(R)$  with  $t'_\ell$  substituted for  $z(L)$  and  $t'_r$  substituted for  $z(R)$  is then the tree  $0 \xrightarrow{a} \frac{1}{\sqrt{2}}$  and, similarly, the result of  $\frac{1}{\sqrt{2}}z(L) - \frac{1}{\sqrt{2}}z(R)$  is the tree  $0 \xrightarrow{a} \frac{1}{\sqrt{2}}$ . Finally,  $t' = 0 \xrightarrow{a} \frac{1}{\sqrt{2}} \xrightarrow{a} \frac{1}{\sqrt{2}}$ .  $\square$

For a tree language  $\mathcal{L} \subseteq \mathbb{T}_\Gamma$ , the *image* of  $\mathcal{L}$  in  $\mathcal{T}$  is the tree language  $\mathcal{T}(\mathcal{L}) = \{\mathcal{T}(t) \mid t \in \mathcal{L}\}$ .

#### 4 Parameterized Verification of Quantum Circuits

Let us now introduce our verification framework. As mentioned before, we use SWTAs to encode sets of quantum states and WTTs to encode quantum gates. Let us now describe the following two approaches to verification of parameterized circuits:

- (1) *Relational verification*: Here, we are given two SWTAs,  $\mathcal{A}_{pre}$  and  $\mathcal{A}_{post}$ , which represent the quantum states in the precondition and in the postcondition respectively, such that the relation (which input quantum state should be mapped to which output quantum state) is captured by the corresponding trees having the same color sequence. The circuit is given by a sequence of WTTs  $\mathcal{T}_1, \dots, \mathcal{T}_k$ . In the verification, we compute the SWTA  $\mathcal{A}_{Result} = \mathcal{T}_k(\mathcal{T}_{k-1}(\dots(\mathcal{T}_1(\mathcal{A}_{pre}))\dots))$  using a sequence of transducer image computation steps (Section 5.7) and then test  $\llbracket \mathcal{A}_{Result} \rrbracket \subseteq \llbracket \mathcal{A}_{post} \rrbracket$  using the algorithm in Section 5.3.
- (2) *Equivalence checking*: We are given two sequences of transducers  $\mathcal{T}_1, \dots, \mathcal{T}_k$  and  $\mathcal{T}'_1, \dots, \mathcal{T}'_l$ . In the verification, we start with the SWTA  $\mathcal{A}_{bases}$  that represents all  $2^n$  computational bases for every size  $n$ . We compute  $\mathcal{A}_{Result} = \mathcal{T}_k(\mathcal{T}_{k-1}(\dots(\mathcal{T}_1(\mathcal{A}_{bases}))\dots))$  and  $\mathcal{A}'_{Result} = \mathcal{T}'_l(\mathcal{T}'_{l-1}(\dots(\mathcal{T}'_1(\mathcal{A}_{bases}))\dots))$  and test  $\llbracket \mathcal{A}_{Result} \rrbracket = \llbracket \mathcal{A}'_{Result} \rrbracket$ . Correctness follows from linearity of the operations and orthogonality of the vectors for the computational bases (from linear algebra, if two unitary operators behave the same on  $2^n$  orthogonal vectors, then they are equal).

Below, we show how this approach can be applied on selected case studies of parameterized circuits.

##### 4.1 The Bernstein-Vazirani Algorithm

In the first example, we show how to use our framework to model a family of circuits implementing the Bernstein-Vazirani (BV) algorithm [10] and verify their functional correctness (the verification will be described in Section 5.3). The family of circuits that we consider is parameterized by  $n \in \mathbb{N}$  such that for every such  $n$ , there will be a circuit with  $n + 1$  qubits (the one additional qubit is an ancilla). Each of the circuits implements the BV algorithm where the secret key is of the form  $(10)^*(1 + \varepsilon) = \{\varepsilon, 1, 10, 101, \dots\}$  such that its length is  $n$  (there is exactly one such a secret for any  $n$ ). The schema for the circuits is given in Fig. 9. We note that the oracle is specialized for the given family of secret keys. The circuit starts,

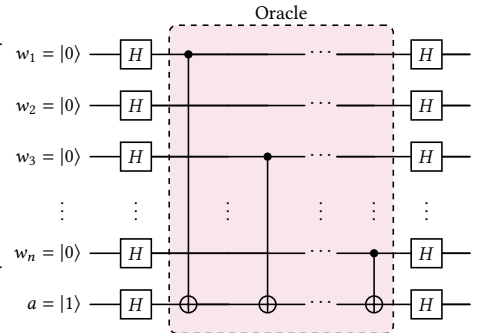


Fig. 9. Circuit implementing the Bernstein-Vazirani algorithm for the secret  $(10)^*(1 + \varepsilon)$

for  $n$  qubits, with the quantum state  $|w_1 \dots w_n a\rangle = |0 \dots 01\rangle$  where  $w_i$  are working qubits and  $a$  is an ancilla, and the secret key will be  $s_1 \dots s_n = 101 \dots$ . The expected output of the circuit is the quantum state  $|s_1 \dots s_n 1\rangle$ , i.e., the working qubits contain the secret key.

We model the problem by providing the following components: (i) SWTAs  $\mathcal{A}_{\text{Pre}}$  and  $\mathcal{A}_{\text{Post}}$  that encode the pre- and post-conditions, (ii) transducer  $\mathcal{T}_{\text{H}^\otimes}$  that represents applying the H gate on all qubits, and (iii) transducer  $\mathcal{T}_{\text{Oracle}}$  representing the oracle. They are defined as follows:

- (1)  $\mathcal{A}_{\text{Pre}}$  is an SWTA that accepts perfect trees where the branch  $0 \dots 01$  has the value 1 and all other branches have the value 0, which can be achieved using the following two transitions (with the root state  $s_1$  and the set of leaf states  $\{s_2\}$ ):  $\{s_1 \xrightarrow{\text{H}} w(s_1, 0s_1), s_1 \xrightarrow{\text{H}} a(0s_2, s_2)\}$ . We emphasize that the symbol  $w$  is used to match *any* of the working qubits  $w_1, \dots, w_n$ .
- (2)  $\mathcal{A}_{\text{Post}}$  is an SWTA that accepts all perfect trees where the branch  $1010 \dots x1$  (with  $x = 0$  for an even  $n$  and  $x = 1$  for an odd  $n$ ) has the value 1 and all other branches have the value zero, which is defined as follows (with the root state  $g$  and the set of leaf states  $\{c\}$ ):

$$\rightarrow g \xrightarrow{\text{H}} w(0h, h) \quad \rightarrow g \xrightarrow{\text{H}} a(0c, c) \quad h \xrightarrow{\text{H}} w(g, 0g) \quad h \xrightarrow{\text{H}} a(0c, c) \quad (12)$$

- (3)  $\mathcal{T}_{\text{H}^\otimes}$  is a WTT with the two transitions (and with  $u$  being both a root and a leaf state)

$$\begin{aligned} \rightarrow u &\rightarrow w\left(\frac{1}{\sqrt{2}}u(L) + \frac{1}{\sqrt{2}}u(R), \frac{1}{\sqrt{2}}u(L) - \frac{1}{\sqrt{2}}u(R)\right) \\ \rightarrow u &\rightarrow a\left(\frac{1}{\sqrt{2}}u(L) + \frac{1}{\sqrt{2}}u(R), \frac{1}{\sqrt{2}}u(L) - \frac{1}{\sqrt{2}}u(R)\right). \end{aligned} \quad (13)$$

We refer the reader to Section 6 for details on how it can be obtained.

- (4)  $\mathcal{T}_{\text{Oracle}}$  is a WTT that models applying a series of CX gates with controls on odd working qubits  $w_{2i+1}$  and the ancilla  $a$  being the target. Our construction of  $\mathcal{T}_{\text{Oracle}}$  is based on the key observation that the ancilla qubit should be flipped only when there is an odd number of 1's in the secret. Therefore, the  $\mathcal{T}_{\text{Oracle}}$ 's states have the form  $r^i$  (going to read a working qubit with secret value 1 or the ancilla qubit) and  $s^i$  (going to read a working qubit with secret value 0 or the ancilla qubit), for  $i \in \{0, 1\}$ , which represents whether the automaton has already seen even (for  $i = 0$ ) or odd (for  $i = 1$ ) number of ones along the tree branch. The state  $r^0$  is the root state and state  $l$  is the only leaf state.

$$\begin{aligned} \rightarrow r^0 &\rightarrow w(s^0(L), s^1(R)) & s^0 &\rightarrow w(r^0(L), r^0(R)) & s^1 &\rightarrow w(r^1(L), r^1(R)) & r^1 &\rightarrow w(s^1(L), s^0(R)) \\ \rightarrow r^0 &\rightarrow a(l(L), l(R)) & s^0 &\rightarrow a(l(L), l(R)) & s^1 &\rightarrow a(l(R), l(L)) & r^1 &\rightarrow a(l(R), l(L)) \end{aligned} \quad (14)$$

We can now establish correctness of the model by computing  $\mathcal{A}_{\text{Result}} = \mathcal{T}_{\text{H}^\otimes}(\mathcal{T}_{\text{Oracle}}(\mathcal{T}_{\text{H}^\otimes}(\mathcal{A}_{\text{Pre}})))$  and testing whether  $\llbracket \mathcal{A}_{\text{Result}} \rrbracket = \llbracket \mathcal{A}_{\text{Post}} \rrbracket$ . An illustration of how  $\mathcal{T}_{\text{H}^\otimes}(\mathcal{T}_{\text{Oracle}}(\mathcal{T}_{\text{H}^\otimes}(\mathcal{A}_{\text{Pre}})))$  is constructed is given in [2].

We note that our choice of a fixed secret family is not the only way of modeling the circuit. Instead, it is possible to model a BV circuit where the secret is arbitrary. In such a case, the secret is a part of the input in which the working qubits are interleaved with secret qubits.

## 4.2 Arithmetic Circuits

Another class of examples to which our approach applies is that of arithmetic circuits, such as adders and comparators. These circuits frequently serve as basic components of quantum algorithms, e.g., within the state preparation stage. In this section, we focus on the verification of the functional correctness of a carry-ripple adder [28] given in Fig. 10c. The adder expects two binary numbers,  $a_1 \dots a_n$  and  $b_1 \dots b_n$ , as well as an initial carry bit  $c_1$  and an ancilla initialized to  $|0\rangle$ , and stores the sum of the numbers and the carry bit to the qubits  $b_1 \dots b_n$ , with the output carry stored to the ancilla (the contents of the  $a_1 \dots a_n$  qubits is unchanged and the output value of  $c_1$  is 0). The circuit is composed of three parts: (i) a “downward staircase” sequence of Majority (MAJ, Fig. 10a) gates, denoted

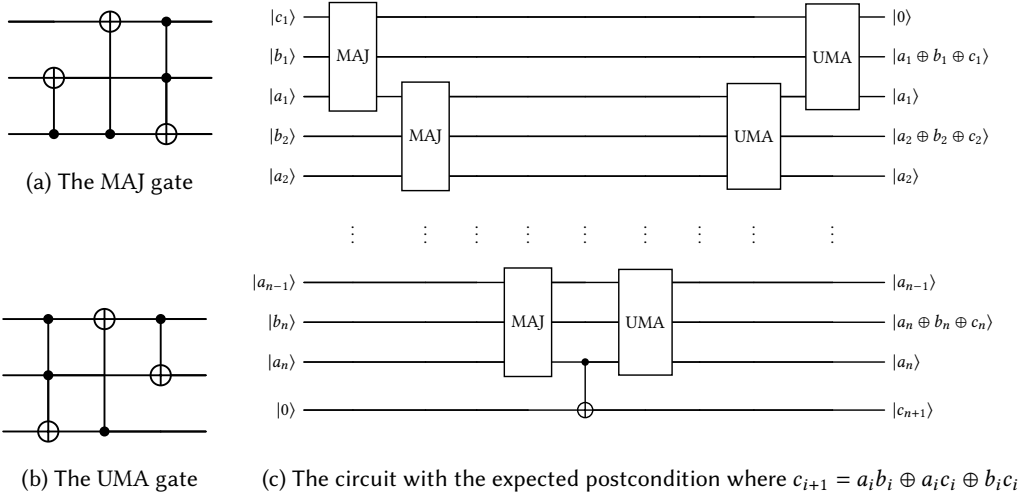


Fig. 10. A circuit implementing a ripple-carry adder.

as  $\text{MAJ}^{\searrow}$ , (ii) a CX between  $a_n$  and the ancilla, and (iii) an “upward staircase” sequence of UnMajority and Add (UMA, Fig. 10b) gates, denoted as  $\text{UMA}^{\nearrow}$ . The precondition can be modelled by an SWTA accepting trees representing all bases with the ancilla 0 and the postcondition can be expressed by an SWTA accepting the expected results (the construction is straightforward but quite tedious).

To verify the functional correctness of the adder using our framework, we need to provide the transducers  $\mathcal{T}_{\text{MAJ}}^{\searrow}$  and  $\mathcal{T}_{\text{UMA}}^{\nearrow}$  for the staircase sequences  $\text{MAJ}^{\searrow}$  and  $\text{UMA}^{\nearrow}$  respectively. We start with constructing the transducer  $\mathcal{T}_{\text{MAJ}}^{\searrow}$  for the MAJ gate (Fig. 10a), which can be easily done by composing the transducers for the CX and CCX gates using the transducer composition algorithm from Section 5.8. Then we need to construct  $\mathcal{T}_{\text{MAJ}}^{\searrow}$  expressing a parameterized sequence of MAJ gates in the given staircase pattern. For this, we use the algorithm from Section 6.2, which takes a (fixed-input size) transducer and transforms it into a transducer that represents the given regular pattern for an arbitrary input size. The transducer  $\mathcal{T}_{\text{UMA}}^{\nearrow}$  is prepared similarly and the verification of the adder then proceeds in the same way as in Section 4.1.

### 4.3 Quantum Error Correction Code

As the next case study, we focus on verification of *quantum error correction codes* (QECC) and consider the *syndrome extraction* circuit of a *repetition code* used to correct bit-flip errors [50]. The code works such that a qubit  $x_1$ , whose value  $\alpha|0\rangle + \beta|1\rangle$  we aim to protect, is entangled using CX gates with qubits  $x_2, \dots, x_n$  (in a similar way as done by the GHZ circuit [33]), obtaining the state  $\varphi = \alpha|0^n\rangle + \beta|1^n\rangle$ . Then  $\varphi$  enters a noisy channel, which may perform some bit flips, so that we obtain a state  $\varphi' = \alpha|w\rangle + \beta|\bar{w}\rangle$  where  $w, \bar{w}$  are binary strings of the length  $n$  and  $\bar{w}$  is the binary complement of  $w$ . The next step is the syndrome extraction, implemented by the circuit in Fig. 11 (for  $n = 4$ ). The circuit uses  $n - 1$  ancillas

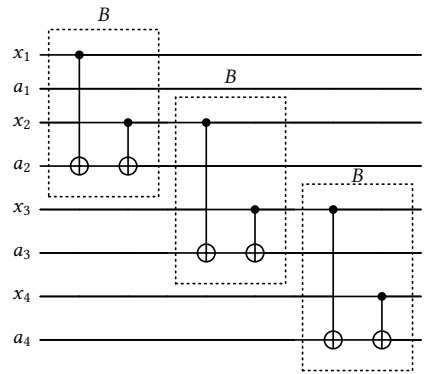


Fig. 11. Syndrome extraction circuit

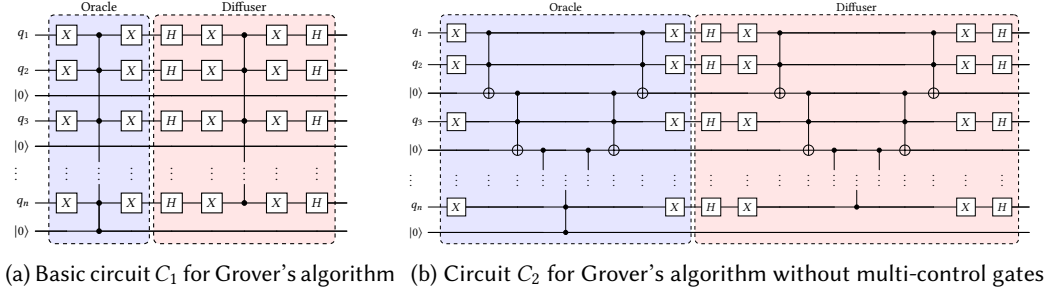


Fig. 12. Two implementations of a single iteration of Grover's algorithm with the solution  $|0^n\rangle$ .

(we use  $n$  in the figure to keep the structure regular;  $a_1$  can be removed) to detect bit-flips such that (i) if a single bit-flip error occurs at  $x_1$ , the ancilla  $a_2$  flips from  $|0\rangle$  to  $|1\rangle$ , (ii) if the error occurs at the qubit  $x_i$  for  $2 \leq i \leq n-1$ , both  $a_i$  and  $a_{i+1}$  flip, and (iii) if the error is at  $x_n$ , then only  $a_n$  is affected. We model the verification of the syndrome extraction circuit (for all sizes) as follows: For the precondition, we consider an SWTA that accepts all trees representing quantum states  $\frac{1}{\sqrt{2}}|w\rangle + \frac{1}{\sqrt{2}}|\bar{w}\rangle$ , where  $w$  contains at most one 1 at an odd position (corresponding to at most one error), and for the postcondition, we construct an SWTA that represents the requirement on the values of the ancillas as described above. The transducer for the circuit is constructed using the algorithm from Section 6.2 and the verification is done similarly as in Section 4.1.

#### 4.4 Amplitude Amplification Circuits

Amplitude amplification algorithms form a fundamental class of quantum search algorithms that may provide speedups (though mostly polynomial) over classical algorithms [12, 13, 34]. Among them, the best-known example is *Grover's algorithm* [34], which operates iteratively: with each iteration, the probability of successfully finding the correct answer increases (until a certain bound). The basic (parameterized) circuit  $C_1$  for one iteration of Grover's algorithm is shown in Fig. 12a.

In practice, the circuit we will use may deviate from the basic form. For instance, we might replace all multi-controlled gates with those supported by a concrete hardware, e.g., with a cascade of CCX gates, in the circuit  $C_2$  in Fig. 12b. In order to verify correctness of such a modification for any circuits in the families, we will leverage the SWTA-based framework in the following way: We will construct the SWTA  $\mathcal{A}_{bases}$  that accepts trees encoding all basis states where ancillas are set to zero (the construction is simple). Then, we perform two computations: (i)  $\mathcal{B}_1 = C_1(\mathcal{A}_{bases})$  and (ii)  $\mathcal{B}_2 = C_2(\mathcal{A}_{bases})$ . One can easily see that all of the required parameterized (and standard) gates can be implemented using WTTs. In order to establish correctness of the second circuit, it is enough to check whether  $\mathcal{B}_1$  and  $\mathcal{B}_2$  are functionally equivalent,  $\llbracket \mathcal{B}_1 \rrbracket = \llbracket \mathcal{B}_2 \rrbracket$ .

#### 4.5 Circuit for Hamiltonian Simulation

*Hamiltonian simulation* is a fundamental computational task in quantum computing, underpinning a broad class of quantum algorithms with applications in physics, chemistry, and material science. At the core is the approximation of the unitary evolution  $e^{-i\mathcal{H}t}$ , where  $\mathcal{H}$  is a Hermitian matrix (the Hamiltonian) describing the dynamics of a quantum system, and  $t$  is a real-valued time parameter. Efficient circuit implementations of this operator are critical for simulating quantum systems.

As an example, we focus on the simulation of a *1D Heisenberg spin chain* [9], a well-studied model whose Hamiltonian is  $\mathcal{H}_{Heis_n} = \sum_{j=1}^{n-1} (X_j X_{j+1} + Y_j Y_{j+1} + Z_j Z_{j+1})$ . The evolution operator  $e^{-i\mathcal{H}_{Heis_n}t}$

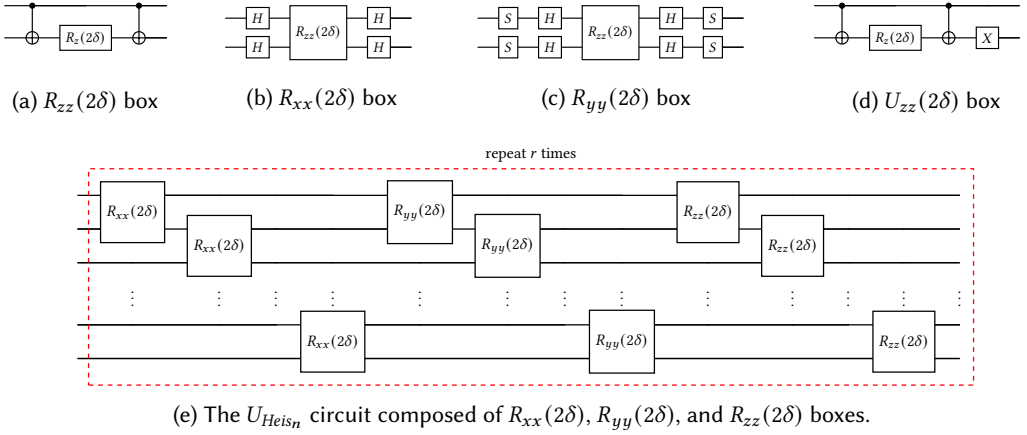


Fig. 13. The circuit implementing the first-order approximation of  $e^{-i\mathcal{H}_{Heis_n}\delta}$ , where  $\delta = \frac{t}{r}$  for some large integer  $r$  that sets the precision of simulation,  $R_z(2\delta) = \begin{bmatrix} e^{-i\delta} & 0 \\ 0 & e^{i\delta} \end{bmatrix}$ . We pick  $t = \pi$  and  $r = 4$  in the experiment.

can be approximated using the parameterized circuit  $U_{Heis_n}$  obtained via a *first-order Trotter-Suzuki decomposition* [47, Sec. 4.7.2] shown in Fig. 13e.

For efficiency, quantum circuits are often further optimized before being executed on hardware. In Fig. 14, we show an optimized version of the original circuit. Our task is to verify that the optimized versions of the circuits are equivalent to the original ones. We test the equivalence in the same way as in Section 4.4. For constructing the necessary transducers for the staircase sequences of boxes, we use the algorithm from Section 6.2.

#### 4.6 Experimental Evaluation

We implemented the techniques described in the paper in a prototype tool named AUTOQ-PARA [3]<sup>5</sup> and used it to evaluate the case studies described in Section 4. The experiments were evaluated on a computer with the Intel i7-1365U CPU and 32 GiB of RAM running Fedora Linux 42. The results are given in Table 1. We are not aware of any other tool supporting fully automated parameterized verification of quantum circuits that we could compare with. From the table, you can see that most of the circuits were verified quickly, with the longest time taken by the adder circuit (Section 4.2). The adder took the longest due to the complexity of the circuit (compared to other circuits) and the corresponding postcondition, which made the constructed SWTAs and WTTs large. Our implementation is an early prototype and there are many opportunities to make it faster.

Table 1. Results of verification of our case studies.

circuit	time
BV	0.014 s
Grover	0.088 s
Adder	11.007 s
QECC	0.314 s
Hamiltonian simulation	0.663 s

### 5 Properties and Operations over SWTAs and WTTs

In this section, we lay out the operations and decision problems for SWTAs and WTTs.

#### 5.1 Domain DFA

Let  $\mathcal{A} = \langle Q, \delta, \Omega, \text{root}, E \rangle$  be an SWTA over  $\Gamma$ . We define the *domain DFA* of  $\mathcal{A}$ , denoted  $\text{dom}(\mathcal{A})$ , to be a *deterministic finite automaton over finite words* (DFA) [30]  $\text{dom}(\mathcal{A}) = \langle P, \Gamma \times \Omega, \Delta, \{\text{root}\}, P_f \rangle$

<sup>5</sup>To ensure accuracy, we restrict our attention to the subset of complex numbers that admit an algebraic representation. See [2] for details.

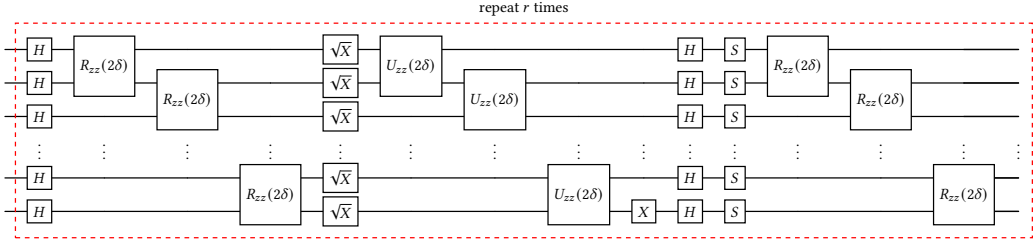


Fig. 14. An optimized version of  $U_{Heis_n}$  obtained using the equalities  $H \cdot H = \text{Id}$ ,  $H \cdot S \cdot S \cdot H = X$  and  $H \cdot S \cdot H = \sqrt{X}$ .

with the set of states  $P = 2^Q$ , the initial state  $\{\text{root}\}$ , final states  $P_f = \{S \in P \mid S \subseteq E \wedge S \neq \emptyset\}$ , and the transition function  $\Delta(S, \langle a, \bullet \rangle) = \bigcup_{s \in S} \{p \in Q \mid s \xrightarrow{\bullet} a(\ell, r) \in \delta \wedge p \in \text{st}(\ell) \cup \text{st}(r)\}$ . The following theorem establishes the connection between  $\text{dom}(\mathcal{A})$  and the domain of  $\llbracket \mathcal{A} \rrbracket$ .

**THEOREM 5.1.** *Let  $\mathcal{A}$  be an SWTA. Then  $\mathcal{L}(\text{dom}(\mathcal{A})) = \text{dom}(\llbracket \mathcal{A} \rrbracket)$ .*

## 5.2 SWTA Language Emptiness

The language emptiness problem for SWTAs asks whether a given SWTA accepts at least one tree.

**THEOREM 5.2.** *Given an SWTA  $\mathcal{A}$ , deciding whether  $\mathcal{L}(\mathcal{A}) = \emptyset$  is PSPACE-complete.*

**PROOF.** (PSPACE-hardness) The hardness follows by reduction from universality of a *nondeterministic finite automaton* (NFA), which is a known PSPACE-complete problem [30]. In particular, consider an NFA  $\mathcal{N} = \langle Q, \Sigma, \Delta, q_I, F \rangle$ , where  $Q$  is a set of states,  $\Sigma$  is the input (finite) alphabet,  $q_I \in Q$  is the initial state (w.l.o.g. we consider exactly one initial state),  $F \subseteq Q$  is the set of final states, and  $\Delta: Q \times \Sigma \rightarrow 2^Q$  is the NFA transition function. We assume that  $\mathcal{N}$  is complete, i.e.,  $\Delta(P, a) \neq \emptyset$  for any  $P \in Q, a \in \Sigma$ . We construct the SWTA  $\mathcal{A}_{\mathcal{N}} = \langle Q, \delta, \Sigma, q_I, Q \setminus F \rangle$  over the alphabet  $\{a\}$ , i.e., with the same states as  $\mathcal{N}$  with a single input symbol  $a$  and the set of colours corresponding to the alphabet  $\Sigma$ , and with a state being a leaf state iff it was not a final state of  $\mathcal{N}$ , where  $\delta$  is defined as  $\delta = \{q \xrightarrow{c} a(x, x) \mid q \in Q, c \in \Sigma, P = \Delta(q, c), x = \sum_{p \in P} p\}$ . Then it holds that if there is a word  $w = b_1 \dots b_n$  not in  $\mathcal{L}(\mathcal{N})$ —which means that the set of reachable states in  $\mathcal{N}$  over  $w$  does not contain any state from  $F$ —then there will be a tree  $t_w$  such that  $\llbracket \mathcal{A}_{\mathcal{N}} \rrbracket(w') = t_w$  (since all reached states will be leaf states) where  $w' = \langle a, b_1 \rangle \dots \langle a, b_n \rangle$ . Therefore,  $\mathcal{N}$  is universal iff  $\mathcal{L}(\mathcal{A}_{\mathcal{N}})$  is empty.

(PSPACE-membership) We reduce the SWTA language emptiness problem to checking emptiness of the language of the domain DFA  $\text{dom}(\mathcal{A})$  (obviously,  $\mathcal{L}(\mathcal{A}) = \emptyset$  iff  $\mathcal{L}(\text{dom}(\mathcal{A})) = \emptyset$ ). While  $\text{dom}(\mathcal{A})$  can be exponentially larger than  $\mathcal{A}$  (its construction is based on the subset construction), we can explore it on the fly during the emptiness check in the standard way, similarly as in NFA universality checking, which can be done in nondeterministic polynomial space. PSPACE membership then follows from Savitch's theorem.  $\square$

## 5.3 Functional Equivalence and Inclusion Checking

Our framework is based on testing functional equivalence and functional inclusion of two SWTAs  $\mathcal{A}$  and  $\mathcal{B}$ , i.e., on checking whether  $\llbracket \mathcal{A} \rrbracket = \llbracket \mathcal{B} \rrbracket$  and  $\llbracket \mathcal{A} \rrbracket \subseteq \llbracket \mathcal{B} \rrbracket$  respectively. Unlike language equivalence and inclusion (i.e.,  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$  and  $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ )—which are undecidable for SWTAs, cf. Section 5.5—their functional counterparts can be decided efficiently and are usable for verification of parameterized quantum circuits.

Checking functional equivalence and inclusion are done via a reduction to solving the *zero-invariant problem of a linear transition system*, which can be done using the so-called Karr's algorithm [24, 37, 46]. Let us begin with defining the necessary concepts. A *linear transition system* (LTS) with  $k$  complex-valued variables arranged in a column vector  $\bar{x} = (x_1, \dots, x_k)^T$  is a tuple  $P = \langle S, s_0, \bar{v}_0, Tr \rangle$ , where  $S$  is a finite set of states,  $s_0 \in S$  is the initial state,  $\bar{v}_0 \in \mathbb{C}^k$  is the (column) vector of initial variable values, and  $Tr$  is a finite set of transitions of the form  $s_1 \xrightarrow{A} s_2$ , where  $s_1, s_2 \in S$  and  $A \in \mathbb{C}^{k \times k}$  is a *linear transformation* represented by a  $k \times k$  complex matrix. A transition  $s_1 \xrightarrow{A} s_2 \in Tr$  means that  $P$  can move from state  $s_1$  to state  $s_2$  while reassigning the contents of variables as  $\bar{x} := A\bar{x}$ . The *zero-invariant problem* for  $P$  and  $S_t \subseteq S$  asks whether for all sequences of transitions  $s_0 \xrightarrow{T_1} s_1 \xrightarrow{T_2} \dots \xrightarrow{T_m} s_m \in Tr^*$  such that  $s_m \in S_t$ , it holds that  $T_m \dots T_2 T_1 \bar{v}_0 = \mathbf{0}$  where  $\mathbf{0}$  denotes the  $k$ -dimensional zero vector. Informally, we are asking whether there is a sequence of transitions from the initial configuration  $\langle s_0, \bar{v}_0 \rangle$  that reaches a state in  $S_t$  with at least one variable having a non-zero value.

**THEOREM 5.3.** *The zero-invariant problem for an LTS with  $n$  states and  $k$  variables can be solved in time  $O(nk^3)$  considering a unit cost of arithmetic operations.*

**PROOF.** The problem can be solved by the so-called Karr's algorithm [37, 46] (we note that Karr's algorithm solves a more general problem of computing *affine relationships* in *affine programs*, i.e., programs where transformations are of the form  $\bar{x} := A\bar{x} + \bar{b}$ ; here we only use  $\bar{b} = \mathbf{0}$ ). Intuitively, Karr's algorithm works by starting in the initial state with the initial vector and propagating the vector along the transitions. At each state, we collect the (at most  $k$ ) linearly independent vectors. After the system saturates, we check that all states in  $S_t$  are only reachable with the vector  $\mathbf{0}$ . Karr's algorithm works in the time  $O(nk^3)$  if arithmetic operations have a unit cost.  $\square$

Let us now show how to build an LTS that corresponds to the given SWTA functional equivalence problem for SWTAs  $\mathcal{A} = \langle Q_a, \delta_a, \Omega, root_a, E_a \rangle$  and  $\mathcal{B} = \langle Q_b, \delta_b, \Omega, root_b, E_b \rangle$  over  $\Gamma$  (w.l.o.g. we assume they use the same set of colors  $\Omega$  such that  $\mathbf{1} \in \Omega$  and that  $Q_a \cap Q_b = \emptyset$ ). Let us first construct an SWTA  $\mathcal{A}_{minus}$  such that for all  $w \in (\Gamma \times \Omega)^*$ , if  $\llbracket \mathcal{A} \rrbracket(w) = \llbracket \mathcal{B} \rrbracket(w)$ , then  $\mathcal{A}_{minus}$  will generate a perfect tree of the height  $|w|$  with all leaves being zero, otherwise it will generate a tree of the same height but with at least one non-zero leaf. Concretely, we define  $\mathcal{A}_{minus} = \langle Q, \delta, \Omega, root, E \rangle$  to be an SWTA over  $\Gamma$  where

- $Q = Q_a \cup Q_b \cup \{root\}$  where  $root \notin Q_a \cup Q_b$ ,
- $E = E_a \cup E_b$ , and
- $\delta = \delta_a \cup \delta_b \cup \{root \xrightarrow{\mathbf{1}} \alpha(root_a - root_b, root_a - root_b)\}$  where  $\alpha$  is any symbol from  $\Gamma$ .

**THEOREM 5.4.** *For all  $w \in \text{dom}(\llbracket \mathcal{A} \rrbracket) \cap \text{dom}(\llbracket \mathcal{B} \rrbracket)$ , it holds that  $\llbracket \mathcal{A}_{minus} \rrbracket(\langle \alpha, \mathbf{1} \rangle w)$  is the tree defined as  $\text{cons}(\alpha, t_{minus}, t_{minus})$  where  $t_{minus} = \llbracket \mathcal{A} \rrbracket(w) - \llbracket \mathcal{B} \rrbracket(w)$ . Moreover,  $\text{dom}(\mathcal{A}_{minus}) = \{\langle \alpha, \mathbf{1} \rangle\} \bullet (\text{dom}(\llbracket \mathcal{A} \rrbracket) \cap \text{dom}(\llbracket \mathcal{B} \rrbracket))$ .*

**COROLLARY 5.5.** *Assume that  $\text{dom}(\llbracket \mathcal{A} \rrbracket) = \text{dom}(\llbracket \mathcal{B} \rrbracket)$ . Then it holds that  $\llbracket \mathcal{A} \rrbracket = \llbracket \mathcal{B} \rrbracket$  iff all trees in  $\mathcal{L}(\mathcal{A}_{minus})$  have only zero-valued leaves.*

To check whether  $\llbracket \mathcal{A} \rrbracket = \llbracket \mathcal{B} \rrbracket$ , it is now enough to proceed as follows:

- (1) Check that  $\text{dom}(\llbracket \mathcal{A} \rrbracket) = \text{dom}(\llbracket \mathcal{B} \rrbracket)$  by testing  $\mathcal{L}(\text{dom}(\mathcal{A})) = \mathcal{L}(\text{dom}(\mathcal{B}))$  (alternatively, for testing functional inclusion, we test  $\mathcal{L}(\text{dom}(\mathcal{A})) \subseteq \mathcal{L}(\text{dom}(\mathcal{B}))$ ). This can be done in PSPACE since we can perform an on-the-fly construction of the product of the domain DFAs  $\text{dom}(\mathcal{A})$  and  $\text{dom}(\mathcal{B})$ , which are both exponential-sized, while looking for a state that is accepting in exactly one of the DFAs. If the equivalence does not hold, return false.
- (2) Construct  $\mathcal{A}_{minus}$  and test whether all trees it accepts have only zero-valued leaves.

To test that all trees accepted by  $\mathcal{A}_{\text{minus}}$  have only zeros in their leaves, we reduce the problem to the zero-invariant problem of the LTS  $P_{\text{minus}}$  defined below. First, we define an auxiliary mapping  $\text{mat}(\mathcal{A}_{\text{minus}}): \Gamma \times \Omega \times \{\text{left}, \text{right}\} \rightarrow \mathbb{C}^{k \times k}$  with  $k = |Q|$  being the number of states in  $\mathcal{A}_{\text{minus}}$  (we assume  $Q = \{q_1, \dots, q_k\}$ ). The mapping represents the transitions of  $\mathcal{A}_{\text{minus}}$  as complex matrices in a similar way as in weighted automata [29]. Let us begin by defining, for a linear form  $\mathbf{x} \in \mathbb{L}_Q$ , the vector  $\text{vec}(\mathbf{x}) = (u_1, \dots, u_k)$  such that  $u_i = \mathbf{x}[q_i]$  if  $q_i \in \text{st}(\mathbf{x})$  and  $u_i = 0$  otherwise. Then, for all  $q \in Q$ , we define the function  $\text{row}(q): \Gamma \times \Omega \times \{\text{left}, \text{right}\} \rightarrow \mathbb{C}^k$  such that

$$\text{row}(q)(a, \mathbf{c}, D) = \begin{cases} \text{vec}(\ell) & \text{if } q \xrightarrow{\mathbf{c}} a(\ell, \mathbf{r}) \in \delta \text{ and } D = \text{left}, \\ \text{vec}(\mathbf{r}) & \text{if } q \xrightarrow{\mathbf{c}} a(\ell, \mathbf{r}) \in \delta \text{ and } D = \text{right}, \text{ and} \\ 0 & \text{otherwise.} \end{cases} \quad (15)$$

We then define  $\text{mat}(\mathcal{A}_{\text{minus}})$  by composing the vectors for  $\text{row}(q)$  for all states  $q \in Q$  as follows:

$$\text{mat}(\mathcal{A}_{\text{minus}})(a, \mathbf{c}, D) = \begin{bmatrix} \text{row}(q_1)(a, \mathbf{c}, D) \\ \vdots \\ \text{row}(q_k)(a, \mathbf{c}, D) \end{bmatrix}. \quad (16)$$

Further, consider the domain DFA  $\text{dom}(\mathcal{A}_{\text{minus}}) = \langle G, \Gamma \times \Omega, \Delta, g_0, G_f \rangle$ . We construct the LTS  $P_{\text{minus}} = \langle S, s_0, \bar{v}_0, Tr \rangle$  in the following way:

- $S = 2^Q \times G$ ,
- $s_0 = \langle \{\text{root}\}, g_0 \rangle$ ,
- $\bar{v}_0 = (1, 0, \dots, 0)$ , while assuming that  $q_1 = \text{root}$ , and
- $\langle U, g \rangle \xrightarrow{a} \langle U', g' \rangle \in Tr$  iff there exist  $a \in \Gamma$  and  $\mathbf{c} \in \Omega$  such that  $g' = \Delta(g, \langle a, \mathbf{c} \rangle)$  and one of the following holds:
  - $U' = \bigcup \{\text{st}(\ell) \mid u \in U, u \xrightarrow{\mathbf{c}} a(\ell, \mathbf{r}) \in \delta\}$  and  $A = \text{mat}(\mathcal{A}_{\text{minus}})(a, \mathbf{c}, \text{left})$  or
  - $U' = \bigcup \{\text{st}(\mathbf{r}) \mid u \in U, u \xrightarrow{\mathbf{c}} a(\ell, \mathbf{r}) \in \delta\}$  and  $A = \text{mat}(\mathcal{A}_{\text{minus}})(a, \mathbf{c}, \text{right})$ .

Intuitively, in each  $P_{\text{minus}}$ 's state  $\langle U, g \rangle$ , the  $U$ -component denotes the set of  $\mathcal{A}_{\text{minus}}$ 's states “active” at a particular branch of the input tree while  $g$  keeps track of  $\mathcal{A}_{\text{minus}}$ 's active states on *all branches* (since in order to accept a tree, all branches need to reach a leaf state in all of their active states). We also construct  $S_t = \{\langle U, g \rangle \in S \mid g \in G_f\}$ , which is a set of states of the LTS representing computations of  $\mathcal{A}_{\text{minus}}$  where all branches reach a leaf node.

**THEOREM 5.6.** *The functional equivalence  $\llbracket \mathcal{A} \rrbracket = \llbracket \mathcal{B} \rrbracket$  holds iff  $\text{dom}(\llbracket \mathcal{A} \rrbracket) = \text{dom}(\llbracket \mathcal{B} \rrbracket)$  and the zero-invariant problem for  $P_{\text{minus}}$  and  $S_t$  holds. Similarly, the functional inclusion  $\llbracket \mathcal{A} \rrbracket \subseteq \llbracket \mathcal{B} \rrbracket$  holds iff  $\text{dom}(\llbracket \mathcal{A} \rrbracket) \subseteq \text{dom}(\llbracket \mathcal{B} \rrbracket)$  and the zero-invariant problem for  $P_{\text{minus}}$  and  $S_t$  holds. Moreover, the two problems can be decided in EXPSpace.*

**PROOF SKETCH.** The correctness can be established using the reasoning outlined above. Focusing on the complexity part, we note that the size of  $P_{\text{minus}}$  is exponential to the size of the input. The length of a sequence of transitions that would lead to a non-zero vector in some of the target states is then also exponential. On the first sight, this might permit a PSPACE algorithm, which would guess the sequence on the fly and locally generate successors of vectors until the requested counterexample is reached. The problem is, however, that the sizes of the bit representations of the numbers in the vector might grow exponentially, so they do not fit into polynomial space. They are, however, bounded by EXPSpace.  $\square$

**THEOREM 5.7.** *The functional equivalence and inclusion problems for SWTAs are PSPACE-hard.*

**PROOF SKETCH.** We prove PSPACE-hardness of the problems by a reduction from SWTA language emptiness, which is PSPACE-complete (cf. Theorem 5.2). Concretely, we can test language emptiness

of an SWTA  $\mathcal{A}$  by constructing an SWTA  $\mathcal{B}$  with an empty language (so, also  $\llbracket \mathcal{B} \rrbracket = \emptyset$ ) and checking whether  $\llbracket \mathcal{A} \rrbracket = \llbracket \mathcal{B} \rrbracket$  or  $\llbracket \mathcal{A} \rrbracket \subseteq \llbracket \mathcal{B} \rrbracket$ .  $\square$

#### 5.4 Language Intersection Emptiness

**THEOREM 5.8.** *Given two SWTAs  $\mathcal{A}$  and  $\mathcal{B}$ , it is undecidable whether  $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B}) = \emptyset$ .*

**PROOF.** By reduction from *Post's correspondence problem* (PCP) [51]. Assume an instance  $I$  of PCP over the alphabet  $\{1, \dots, 9\}$  with the set of  $k$  pairs  $I = \{\langle \alpha_1, \beta_1 \rangle, \dots, \langle \alpha_k, \beta_k \rangle\}$ . For a word  $w$ , let  $\text{rev}(w)$  denote the reverse of  $w$ ,  $\text{num}(w)$  denote the numerical value of  $w$ , with  $\text{num}(\epsilon) = 0$ , and  $|w|$  denote the length of  $w$ . Let us now construct the SWTA  $\mathcal{A} = \langle Q, \delta, \Omega, q, E \rangle$  over the alphabet  $\{a\}$  as follows:

- $Q = \{q, p, r, s, u\}$ ,  $E = \{p, r, s, u\}$ ,  $\Omega = \{\textcolor{red}{1}, \dots, \textcolor{black}{k}\}$ , and
- $\delta = \delta_1 \cup \delta_2 \cup \delta_\alpha \cup \delta_\beta$  where
  - $\delta_1 = \{q \xrightarrow{\textcolor{red}{1}} a(p - r, 0s)\}$ ,
  - $\delta_2 = \{s \xrightarrow{\textcolor{black}{c}} a(s, s), u \xrightarrow{\textcolor{black}{c}} a(u, 0s) \mid \textcolor{black}{c} \in \Omega\}$ ,
  - $\delta_\alpha = \{p \xrightarrow{\textcolor{black}{c}} a(10^{|\alpha_c|} \cdot p + \text{num}(\text{rev}(\alpha_c)) \cdot u, 0s) \mid \langle \alpha_c, \beta_c \rangle \in I\}$ , and
  - $\delta_\beta = \{r \xrightarrow{\textcolor{black}{c}} a(10^{|\beta_c|} \cdot r + \text{num}(\text{rev}(\beta_c)) \cdot u, 0s) \mid \langle \alpha_c, \beta_c \rangle \in I\}$ .

Intuitively, we are trying to construct a tree where the leftmost branch leads to the leaf with value 0 (by construction, all other branches always go to zero-valued leaves) iff the PCP has a solution. The concatenation in the PCP is simulated by addition and digit-shifting implemented by multiplication by a power of 10. Therefore, if the PCP has a solution, then (and only then) the language of  $\mathcal{A}$  contains a tree whose all leaves are valued 0.

Moreover, let  $\mathcal{B}$  be an SWTA accepting all perfect binary trees of the height  $\geq 1$  with leaves labeled by 0 (such an SWTA needs just two transitions  $v \xrightarrow{\textcolor{red}{1}} a(0z, 0z)$  and  $z \xrightarrow{\textcolor{red}{1}} a(z, z)$  with the root state  $v$  and the set of leaf states  $\{z\}$ ). It holds that  $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B}) \neq \emptyset$  iff  $I$  has a solution.  $\square$

#### 5.5 Language Equivalence and Inclusion

In this section, we show that questions about language inclusion or equivalence of a pair of SWTAs are undecidable.

**THEOREM 5.9.** *Given SWTAs  $\mathcal{A}$  and  $\mathcal{B}$ , the following questions are undecidable:*

- (1)  $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$  and
- (2)  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ .

**PROOF SKETCH.** Our proof of the undecidability of  $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$  is done by a reduction from the emptiness problem of a Turing machine (TM)  $M$ , inspired by the proof of undecidability of universality of timed automata [6, Theorem 5.2]. In particular, we construct SWTAs  $\mathcal{A}$  and  $\mathcal{B}$  over the alphabet  $\{a\}$  such that  $\mathcal{L}(M) \neq \emptyset \Leftrightarrow \mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B})$ .  $\mathcal{B}$  will be constructed as an SWTA that accepts all perfect trees with exactly one 1-valued leaf node and 0-valued leaves elsewhere (the construction is simple).

On the other hand,  $\mathcal{A}$  will be constructed in the following way. Let  $\rho$  be a run (i.e., a sequence of configurations) of  $M$ , then we use  $\text{enc}(\rho)$  to denote an encoding of  $\rho$  into a binary string (done in the standard way). Our goal is to construct  $\mathcal{A}$  such that every tree accepted by  $\mathcal{A}$  has all leaves labelled by 0 except the leaf at the end of one significant branch, which will have the value 1 iff the branch is *not* an encoding of an accepting run of  $M$  (and it will have any other value otherwise). In other words, if there exists an accepting run  $\rho_{\text{acc}}$  of  $M$ , then  $\mathcal{L}(\mathcal{A})$  contains a tree whose branch  $\text{enc}(\rho_{\text{acc}})$  does not end with 1 and, therefore,  $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B})$ .

The construction of  $\mathcal{A}$  is technical, but the main idea is that we will start a parallel computation from two states (using a linear form  $p - r$ ) where  $p$  will try to detect a simple error in the given input, such as problems with the encoding or a wrong format of the configurations (which can be described using a regular language, so it is easy to implement them in an SWTA). For detecting harder errors, such as an inconsistency in the encoding of two consecutive configurations, we use the run from  $p$  to guess the position of the error and then use color-based synchronization to communicate with the run from  $r$  the nature of the error;  $r$  will then make sure that the error was guessed correctly. Marking the position can be done by, e.g., multiplying the children states with 2.

The proof of undecidability of  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$  can be done by reduction from the inclusion:  $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B}) \iff \mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{B})$ .  $\square$

## 5.6 Boolean Operations

We will show that SWTAs are closed under language union but are not closed under language intersection and language complement.

**THEOREM 5.10.** *Let  $\mathcal{A}$  and  $\mathcal{B}$  be SWTAs. Then one can effectively construct an SWTA  $\mathcal{C}$  such that  $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{B})$ .*

**PROOF.** Let  $\mathcal{A} = \langle Q_a, \delta_a, \Omega_a, \text{root}_a, E_a \rangle$  and  $\mathcal{B} = \langle Q_b, \delta_b, \Omega_b, \text{root}_b, E_b \rangle$  be SWTAs over  $\Gamma$  such that, w.l.o.g.,  $Q_a \cap Q_b = \emptyset$  and  $\Omega_a \cap \Omega_b = \emptyset$ . Then let  $\mathcal{C} = \langle Q_c, \delta_c, \Omega_c, \text{root}_c, E_c \rangle$  be an SWTA over  $\Gamma$  constructed as follows:

- $Q_c = Q_a \cup Q_b \cup \{\text{root}_c\}$  with  $\text{root}_c \notin Q_a \cup Q_b$ ,
- $\Omega_c = \Omega_a \cup \Omega_b$ ,
- $E_c = E_a \cup E_b \cup G$  where  $G = \{\text{root}_c\}$  if  $\{\text{root}_a, \text{root}_b\} \cap (E_a \cup E_b) \neq \emptyset$ , else  $G = \emptyset$ , and
- $\delta_c = \delta_a \cup \delta_b \cup \{\text{root}_c \xrightarrow{\alpha} \alpha(\ell, r) \mid \text{root}_a \xrightarrow{\alpha} \alpha(\ell, r) \in \delta_a \vee \text{root}_b \xrightarrow{\alpha} \alpha(\ell, r) \in \delta_b\}$ .

One can prove  $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{B})$  in the standard way.  $\square$

**THEOREM 5.11.** *SWTAs are not closed under language intersection and language complement.*

**PROOF.** The non-closure under language intersection is a direct corollary of Theorem 5.2 (language intersection emptiness is undecidable) and Theorem 5.2 (language emptiness is decidable).

For the non-closure under complement, it is easy to notice that the languages of SWTAs are all of a countable size, while the set of all perfect trees with complex-valued leaves is uncountable. An alternative proof can be obtained by assuming (for the sake of contradiction) closure under complement and then, from Theorem 5.10 and Theorem 5.2 and using De Morgan's laws, one would obtain decidability of the intersection emptiness, which contradicts Theorem 5.8.  $\square$

## 5.7 Transducer Image

In the section, we will give an algorithm for computing an SWTA  $\mathcal{T}_2(\mathcal{A}_1)$  representing the language  $\mathcal{T}_2(\mathcal{L}(\mathcal{A}_1))$  where  $\mathcal{A}_1 = \langle Q_1, \delta_1, \Omega, \text{root}_1, E_1 \rangle$  is an SWTA over  $\Gamma$  and  $\mathcal{T}_2 = \langle Q_2, \delta_2, \text{root}_2, E_2 \rangle$  is a WTT over  $\Gamma$ . First, we introduce some useful notation. Let  $q(d) \in Q_2(\mathbb{L}, \mathbb{R})$  and  $\ell, r \in \mathbb{L}_{Q_1}$  (we recall that  $Q_2(\mathbb{L}, \mathbb{R})$  represents the set of ground terms of the form  $q(\mathbb{L}, \mathbb{R})$  for  $q \in Q_2$ ), then we use  $q(d)(\ell, r)$  to denote the linear form over  $Q_1 \times Q_2$  defined as

$$q(d)(\ell, r) = \begin{cases} \sum_{p \in \text{st}(\ell)} \ell[p] \cdot \langle p, q \rangle & \text{if } d = \mathbb{L}, \\ \sum_{p \in \text{st}(r)} r[p] \cdot \langle p, q \rangle & \text{if } d = \mathbb{R}. \end{cases} \quad (17)$$

For instance,  $q(\mathbb{L})(ap_a + bp_b, 0) = a\langle p_a, q \rangle + b\langle p_b, q \rangle$ . We extend the notation to linear forms  $x \in \mathbb{L}_{Q_2(\mathbb{L}, \mathbb{R})}$  such that  $x(\ell, r)$  is obtained from  $x$  by substituting every occurrence of  $q(d)$  in  $x$  by  $q(d)(\ell, r)$  and multiplying and summing up the coefficients to obtain a linear form over  $Q_1 \times Q_2$ .

*Example 5.12.* Consider the linear form  $\mathbf{x} \in \mathbb{L}_{Q_2(\mathbf{L}, \mathbf{R})}$  and linear forms  $\ell, \mathbf{r} \in \mathbb{L}_{Q_1}$  such that

$$\mathbf{x} = \frac{1}{\sqrt{2}}q(\mathbf{L}) + q(\mathbf{R}) - 3s(\mathbf{R}), \quad \ell = p + 0u, \quad \text{and} \quad \mathbf{r} = -2p + \frac{1}{\sqrt{2}}u. \quad (18)$$

Then

$$\begin{aligned} \mathbf{x}(\ell, \mathbf{r}) &= \frac{1}{\sqrt{2}}\langle p, q \rangle + 0\langle u, q \rangle - 2\langle p, q \rangle + \frac{1}{\sqrt{2}}\langle u, q \rangle + 6\langle p, s \rangle - \frac{3}{\sqrt{2}}\langle u, s \rangle \\ &= \frac{1-2\sqrt{2}}{\sqrt{2}}\langle p, q \rangle + \frac{1}{\sqrt{2}}\langle u, q \rangle + 6\langle p, s \rangle - \frac{3}{\sqrt{2}}\langle u, s \rangle, \end{aligned} \quad (19)$$

which is the resulting linear form.  $\square$

The image of  $\mathcal{A}_1$  w.r.t.  $\mathcal{T}_2$  is then the SWTA  $\mathcal{T}_2(\mathcal{A}_1) = \mathcal{A}_3 = \langle Q_3, \delta_3, \Omega, \text{root}_3, E_3 \rangle$  where the components are defined in the following way:

- $Q_3 = Q_1 \times Q_2$ ,  $\text{root}_3 = \langle \text{root}_1, \text{root}_2 \rangle$ ,  $E_3 = E_1 \times E_2$ , and
- $\delta_3 = \{ \langle q_1, q_2 \rangle \xrightarrow{\mathbf{a}} a(\ell_2(\ell_1, \mathbf{r}_1), \mathbf{r}_2(\ell_1, \mathbf{r}_1)) \mid q_1 \xrightarrow{\mathbf{a}} a(\ell_1, \mathbf{r}_1) \in \delta_1, q_2 \rightarrow a(\ell_2, \mathbf{r}_2) \in \delta_2 \}$ .

An example illustrating the computation of a transducer image can be found in [2].

**THEOREM 5.13.**  $\mathcal{L}(\mathcal{A}_3) = \mathcal{T}_2(\mathcal{L}(\mathcal{A}_1))$ .

## 5.8 Transducer Composition

Below, we give a construction of a transducer  $\mathcal{T}_\circ = \mathcal{T}_2 \circ \mathcal{T}_1$  representing the *composition* of functions denoted by transducers  $\mathcal{T}_1 = \langle Q_1, \delta_1, \text{root}_1, E_1 \rangle$  and  $\mathcal{T}_2 = \langle Q_2, \delta_2, \text{root}_2, E_2 \rangle$  over  $\Gamma$ . We first extend the notation from the previous section such that for  $q(s) \in Q_2(\mathbf{L}, \mathbf{R})$  with  $s \in \{\mathbf{L}, \mathbf{R}\}$  and  $\ell, \mathbf{r} \in \mathbb{L}_{Q_1(\mathbf{L}, \mathbf{R})}$ , we use  $q(s)(\ell, \mathbf{r})$  to denote the linear form over  $(Q_1 \times Q_2)(\mathbf{L}, \mathbf{R})$  defined as

$$q(s)(\ell, \mathbf{r}) = \begin{cases} \sum_{\langle p, \mathbf{L} \rangle \in \text{st}(\ell)} \ell[p] \cdot \langle p, q \rangle(\mathbf{L}) + \sum_{\langle p, \mathbf{R} \rangle \in \text{st}(\ell)} \ell[p] \cdot \langle p, q \rangle(\mathbf{R}) & \text{if } s = \mathbf{L}, \\ \sum_{\langle p, \mathbf{L} \rangle \in \text{st}(\mathbf{r})} \mathbf{r}[p] \cdot \langle p, q \rangle(\mathbf{L}) + \sum_{\langle p, \mathbf{R} \rangle \in \text{st}(\mathbf{r})} \mathbf{r}[p] \cdot \langle p, q \rangle(\mathbf{R}) & \text{if } s = \mathbf{R}. \end{cases} \quad (20)$$

For instance,  $q(\mathbf{L})(ap_a(\mathbf{L}) + bp_b(\mathbf{R}), 0) = a\langle p_a, q \rangle(\mathbf{L}) + b\langle p_b, q \rangle(\mathbf{R})$ . Similarly as in the previous section, we extend the notation from states  $q \in Q_2$  to linear forms  $\mathbf{x} \in \mathbb{L}_{Q_2(\mathbf{L}, \mathbf{R})}$  such that  $\mathbf{x}(\ell, \mathbf{r})$  is obtained from  $\mathbf{x}$  by substituting every occurrence of  $q(s)$  in  $\mathbf{x}$  by  $q(s)(\ell, \mathbf{r})$  and multiplying and summing up the coefficients to obtain a linear form over  $(Q_1 \times Q_2)(\mathbf{L}, \mathbf{R})$ .

The composition  $\mathcal{T}_2 \circ \mathcal{T}_1$  is then the transducer  $\mathcal{T}_\circ = \langle Q_\circ, \delta_\circ, \text{root}_\circ, E_\circ \rangle$  with its components defined as follows:

- $Q_\circ = Q_1 \times Q_2$ ,  $\text{root}_\circ = \langle \text{root}_1, \text{root}_2 \rangle$ ,  $E_\circ = E_1 \times E_2$ , and
- $\delta_\circ = \{ \langle q_1, q_2 \rangle \rightarrow a(\ell_2(\ell_1, \mathbf{r}_1), \mathbf{r}_2(\ell_1, \mathbf{r}_1)) \mid q_1 \rightarrow a(\ell_1, \mathbf{r}_1) \in \delta_1, q_2 \rightarrow a(\ell_2, \mathbf{r}_2) \in \delta_2 \}$ .

As usual,  $\mathcal{T}_\circ$  constructed by the procedure above may contain some unreachable states and transitions; we therefore use a reachability-based construction starting from  $\langle \text{root}_1, \text{root}_2 \rangle$  in the standard way. An example showing how to compute a transducer composition can be found in [2].

**THEOREM 5.14.** For all trees  $t \in \mathbb{T}_\Gamma$  and WTTs  $\mathcal{T}_1$  and  $\mathcal{T}_2$  over  $\Gamma$ , it holds that  $(\mathcal{T}_2 \circ \mathcal{T}_1)(t) = \mathcal{T}_2(\mathcal{T}_1(t))$ .

## 6 Transducers for Quantum Gates

In this section, we provide constructions of the transducers for a broad class of quantum gates, including those forming a universal gate set and quantum Fourier transform (QFT). Building on the composition algorithm introduced in Section 5.8, we construct the transducer for a fixed-size circuit—referred to as a *box*—and further present an algorithm systematically generating a transducer for size-parameterized circuit families of a specified repetition pattern. Importantly, the expressive power of transducers extends beyond such structured design: they are capable of describing more general families of circuits. For example, as shown in Section 4.1, the transducer of the Bernstein-Vazirani (BV) circuit cannot be obtained through this pattern-based instantiation alone.

### 6.1 Transducers for Atomic Quantum Gates

We assume that a quantum circuit is operating on  $m$  qubits, labeled in order as  $x_1, \dots, x_m$ . The two main types of atomic quantum gates used in state-of-the-art quantum computers are single-qubit gates and (multi-)controlled gates. In general, a single-qubit gate is represented as a *unitary complex matrix*  $U = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ . A controlled gate  $CU$  uses another quantum gate  $U$  as its parameter.  $CU$  consists of a control qubit  $x_i$  and the gate  $U$  is applied only when the control qubit  $x_i$  has value 1. In this section, we construct the transducers for a broad class of quantum gates, including those forming a universal gate set and quantum Fourier transform (QFT), operating on a quantum system of a fixed size  $m$ . Let us fix a single-qubit gate  $U = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ .

*Single Qubit Gates.* The transducer  $\mathcal{T}_{U,i,m}$  for a unitary  $U$  operating on qubit  $x_i$  in a circuit with  $m$  qubits is constructed as  $\mathcal{T}_{U,i,m} = \langle Q, \delta, q_0, E \rangle$  over the alphabet  $\Gamma = \{x_1, \dots, x_m\}$  where  $Q = \{q_j \mid 0 \leq j \leq m\}$ ,  $E = \{q_m\}$ , and  $\delta$  consists of the following transitions:

$$q_j \rightarrow x_{j+1}(q_{j+1}(\text{L}), q_{j+1}(\text{R})) \quad \text{for } j \neq i-1, j < m \quad (21)$$

$$q_{i-1} \rightarrow x_i(aq_i(\text{L}) + bq_i(\text{R}), cq_i(\text{L}) + dq_i(\text{R})) \quad (22)$$

Intuitively, we have a number of states  $q_j$ , which are used to count from 0 to  $m$  as shown in Eq. (21). The effect of applying  $U$  at  $q_{i-1}$  is captured as Eq. (22). This concludes the description of the transducer  $\mathcal{T}_{U,i,m}$ . In the case where  $U = I$  is the identity matrix, the transducer just corresponds to a wire and the index  $i$  does not matter. In this case we simply write  $\mathcal{T}_{I,m}$ .

*Multi-controlled Gates.* Let  $i < j \leq m$ . We will explain our construction of the transducer  $\mathcal{T}_{C_j U_i, m}$  for the controlled- $U$  gate  $C_j U_i$  with control qubit  $x_j$  and target qubit  $x_i$  in a circuit with  $m$  qubits. Let  $\mathcal{T}_1 = \mathcal{T}_{I,m}$  and  $\mathcal{T}_2 = \mathcal{T}_{U,i,m}$ . Observe that, after applying  $C_j U_i$ , the 0-subtrees below  $x_j$  remain the same but the 1-subtrees below  $x_j$  are updated to the corresponding ones after applying  $U$  to qubit  $x_i$ . Let us first define the operation  $\text{zeroL}_j$  on WTTs. The operation changes the input transducer such that the resulting transducer modifies the leaves of 0-branches below  $x_j$  to zero in the image of the transducer. Formally, given a transducer  $\mathcal{T}$  for a single-qubit gate,  $\text{zeroL}_j(\mathcal{T})$  is obtained by taking  $\mathcal{T}$  and replacing the transition

$$q \rightarrow x_j(\ell_1(\text{L}) + \mathbf{r}_1(\text{R}), \ell_2(\text{L}) + \mathbf{r}_2(\text{R})), \quad (23)$$

which acts on the qubit  $x_j$ , by the transition

$$q \rightarrow x_j(\text{zero}_{\ell_1}(\text{L}) + \text{zero}_{\mathbf{r}_1}(\text{R}), \ell_2(\text{L}) + \mathbf{r}_2(\text{R})). \quad (24)$$

Given a linear form  $\mathbf{v}$ , the linear form  $\text{zero}_{\mathbf{v}}$  is obtained from  $\mathbf{v}$  by modifying all its coefficients to zero. All transitions of  $\mathcal{T}$  occurring over other symbols are retained. Note that we can similarly define  $\text{zeroR}_j$ , which modifies the coefficients of the right-hand child. We set  $\mathcal{T}'_1 = \text{zeroR}_j(\mathcal{T}_1)$  and  $\mathcal{T}'_2 = \text{zeroL}_j(\mathcal{T}_2)$ . Finally, we construct

$$\mathcal{T}_{C_j U_i, m} = \mathcal{T}'_2 + \mathcal{T}'_1 = \text{zeroL}_j(\mathcal{T}_{U,i,m}) + \text{zeroR}_j(\mathcal{T}_{I,m}). \quad (25)$$

Here, addition of transducers is an operation that produces a transducer  $\mathcal{T}_a + \mathcal{T}_b$ , such that for a tree  $t$ , it holds that  $(\mathcal{T}_a + \mathcal{T}_b)(t) = \mathcal{T}_a(t) + \mathcal{T}_b(t)$ . Formally,  $\mathcal{T}_a + \mathcal{T}_b$  is defined as follows. Let  $\mathcal{T}_a = (Q_a, \delta_a, \text{root}_a, E_a)$  and  $\mathcal{T}_b = (Q_b, \delta_b, \text{root}_b, E_b)$  be WTTs over  $\Gamma$  where  $Q_a$  and  $Q_b$  are disjoint. Let  $t_a = \text{root}_a \rightarrow x_1(\ell_1(\text{L}) + \mathbf{r}_1(\text{R}), \mathbf{r}_2(\text{R}) + \ell_2(\text{L})) \in \delta_a$  and  $t_b = \text{root}_b \rightarrow x_1(\ell'_1(\text{L}) + \mathbf{r}'_1(\text{R}), \mathbf{r}'_2(\text{R}) + \ell'_2(\text{L})) \in \delta_b$ . Then  $\mathcal{T}_a + \mathcal{T}_b = \mathcal{T}_{C_j U_i, m} = (Q, \delta, \text{root}, E)$  where  $Q = (Q_a \cup Q_b \cup \{\text{root}\}) \setminus \{\text{root}_a, \text{root}_b\}$ ,  $E = E_a \cup E_b$ , and  $\delta$  contains all transitions in  $(\delta_a \cup \delta_b) \setminus \{t_a, t_b\}$  and also the transition  $\text{root} \rightarrow x_1((\ell_1 + \ell'_1)(\text{L}) + (\mathbf{r}_1 + \mathbf{r}'_1)(\text{R}), (\mathbf{r}_2 + \mathbf{r}'_2)(\text{R}) + (\ell_2 + \ell'_2)(\text{L}))$ .

The construction can be directly generalized to multi-controlled gates. For example,

$$\mathcal{T}_{C_k C_j U_i, m} = \text{zeroL}_k(\mathcal{T}_{C_j U_i, m}) + \text{zeroR}_k(\mathcal{T}_{I,m}) = \text{zeroL}_k(\text{zeroL}_j(\mathcal{T}_{U,i,m}) + \text{zeroR}_j(\mathcal{T}_{I,m})) + \text{zeroR}_k(\mathcal{T}_{I,m}).$$

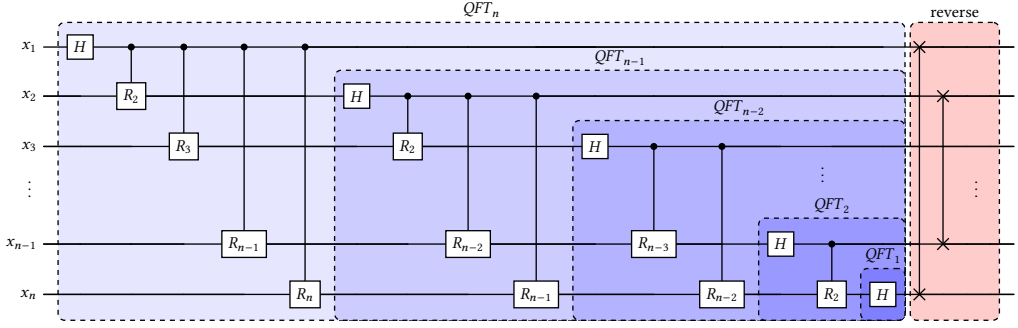


Fig. 15. The QFT circuit, where  $R_k = \begin{pmatrix} 1 & 0 \\ 0 & \gamma_k \end{pmatrix}$  and  $\gamma_k = \omega^{2^{n-k}}$  with  $\omega = e^{\frac{2\pi i}{n}}$ .

**THEOREM 6.1.** *The transducers for single-qubit gates and multi-controlled gates are semantically correct. Moreover, each transducer can be constructed using  $O(m)$  states and  $O(m)$  transitions, where  $m$  is the number of qubits in the circuit.*

**Quantum Fourier Transform Gate.** Given the central role of the Quantum Fourier Transform (QFT), given in Fig. 15, in many quantum algorithms, we explicitly discuss the construction of the transducer for the standard circuit implementation of the  $n$ -qubit QFT gate. The construction can be derived directly using the transducers for atomic gates and composition procedure described in Section 5.8. The construction is detailed in [2], here we provide a short overview to highlight its structure and efficiency. The transducer for  $QFT_{[1..n]}$  acting on a sequence of  $n \leq m$  contiguous qubits requires  $O(n^2)$  states and transitions and to generalize to the full  $m$ -qubit system setting, we append  $O(m)$  additional states and transitions in order to propagate qubits outside the QFT range unchanged. Overall, the transducer has  $O(n^2 + m)$  states and  $O(n^2 + m)$  transitions. We emphasize that the transducer is obtained via the standard composition construction, which has an exponential upper bound on the size of the output (because it is, essentially, a product construction and we are applying it  $O(n^2)$  times). The reason for the only quadratic size of the result comes from the structure of the actual transducers being composed.

Intuitively, the transducer for QFT is built from basic components, transducers representing gates with a parametric number of qubits: the transducer  $\mathcal{T}_{H_i}$  models the Hadamard gate applied on the qubit  $x_i$ , the transducer  $\mathcal{T}_{R_i}$  models the ‘controlled-multi-rotation gate’—in the figure, it represents the chain of gates  $R_2, \dots, R_{n+1-i}$  in the  $i$ -th blue ‘box’ from the left that are applied on qubits  $x_{1+i}, \dots, x_n$  respectively and controlled by qubit  $x_i$ . The transducer representing the whole QFT circuit is now obtained by composing these basic blocks from left to right as shown in the figure, as the transducer  $((((\mathcal{T}_{H_1} \circ \mathcal{T}_{R_1}) \circ \mathcal{T}_{H_2}) \circ \mathcal{T}_{R_2}) \circ \dots \circ \mathcal{T}_{H_n})$ . We do not model the reversion, the right-most red box, in the transducer, as we find it easier to include the reversion into the specification of the verification post-condition. Modelling the reversion would result in an exponential blow-up in the size of the transducer.

**Remark 6.2.** In our use cases, the obtained trasducers were all of a tractable size. The transducer composition construction in the worst case, however, yields a transducer of a quadratic size, so the size of a transducer representing the whole circuit might be exponential to the depth of a circuit. One can see this, e.g., when writing a transducer modelling reversal (as done, e.g., in the case of QFT, cf. Fig. 15), the size of which is exponential (reversion is a purely combinatorial operation that performs a given permutation on the leaves of tree).

## 6.2 Constructing Size-Parameterized Transducers

In this section, we describe our algorithm for systematically generating the transducer from a fixed-size circuit—called a *box*—for a size-parameterized circuit family with a specific repetition pattern.

To illustrate this process step by step, we begin with a concrete example from Fig. 10c, where the circuit consists of a sequence of MAJ boxes. We first build the transducer for this fixed-size MAJ circuit using the transducer composition procedure from Section 5.8, and then generalize it by uniformly removing subscripts from wire labels—for example, replacing  $x_i$  with  $x$ —to indicate that variables now serve as symbolic placeholders rather than fixed wire indices. Wires not involved in active gates are treated as carrying the identity operation, denoted as *Id* (see Fig. 16; for clarity, we retain subscripted indices in the figure). The transducer for *Id* consists of a single state  $\iota$  (both a root and a leaf state) and the transition  $\iota \rightarrow x(\iota, \iota)$ .

A key structural property of this circuit is that each qubit is acted upon by at most two non-identity boxes. Moreover, different qubits can exhibit equivalent transducer behavior. For example, qubits  $x_3$  and  $x_5$  are each processed by two instances of the same MAJ transducer (both working with the same set of states). Suppose that, when reading qubit  $x_3$ , the composition of the transducers (including  $\text{MAJ}_1$  and  $\text{MAJ}_2$ ) is in the state  $\langle \dots, \iota, q_1, s_1, \iota, \dots \rangle$ , and when reading  $x_5$ , it is in the same configuration except one additional  $\iota$  precedes  $q_1$ . The effect on both wires is identical, indicating that the number of surrounding  $\iota$  states does not influence the computation. We can therefore abstract such configurations using the finite tuple  $\langle q_1, s_1 \rangle$ . Although this observation may appear trivial—since composing identity with itself yields identity—it plays a crucial role in our construction. Once an active box becomes inactive, it behaves like  $\iota$ , and the relevant state tuple shifts rightwards. This shifting mechanism governs how the transducer evolves as the circuit grows, and is fundamental to our parameterized transducer construction.

We illustrate the idea behind our construction using the MAJ-gate cascade from Fig. 16. Let  $\text{MAJ}$  be represented by a transducer with a root state  $a$ , leaf state  $\iota$ , and the following transitions:

$$\begin{aligned}
 & \rightarrow a \rightarrow x(b(L) + c(R), e(L) + d(R)) & f \rightarrow x(\iota(L), 0\iota(R)) & k \rightarrow x(0\iota(R), \iota(L)) \\
 & b \rightarrow x(f(L), f(R)) & d \rightarrow x(f(L), k(R)) & g \rightarrow x(\iota(R), 0\iota(L)) & \iota \rightarrow x(\iota(L), \iota(R)) \\
 & c \rightarrow x(h(R), h(L)) & e \rightarrow x(h(R), g(L)) & h \rightarrow x(0\iota(L), \iota(R))
 \end{aligned} \tag{26}$$

Notice that the transducer contains the state  $\iota$  and transition  $\iota \rightarrow x(\iota(L), \iota(R))$  from the *Id* transducer—this is our way of implementing the sequence of *Id* gates below the box (we give formal requirements that the input transducer needs to satisfy later).

We now describe how to compute the transducer for the entire size-parameterized circuit. Our product construction starts in the state  $\langle a \rangle$ , representing the circuit acting on the first qubit. The transition from  $\langle a \rangle$  is the same as the one of  $\text{MAJ}$  from  $a$ , just with states decorated by  $\langle \cdot \rangle$  to be formally a one-tuple (since there is just one transducer active at this point):

$$\langle a \rangle \rightarrow x(\langle b \rangle(L) + \langle c \rangle(R), \langle e \rangle(L) + \langle d \rangle(R)). \tag{27}$$

Consider next the state  $\langle b \rangle$ . As this point, one should determine whether the cascade continues with another MAJ or ends here (since the endpoints of the transition from  $\langle b \rangle$  will need to either be pairs of states from the MAJ transducer or just one state from MAJ that finishes reading out the

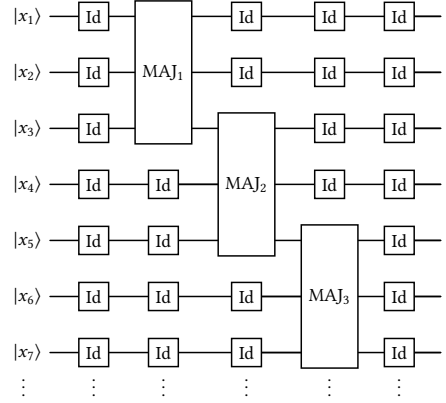


Fig. 16. Parameterized MAJ example

**Algorithm 1:** Construction of Size-Parameterized Transducers

**Input:** A transducer  $\mathcal{T} = (Q, \delta, \text{root}, E)$  over  $\Gamma$  satisfying conditions in the text  
 a number  $n > 0$  denoting the offset of boxes  
 $d \in \{\text{left}, \text{right}\}$  indicating the direction in which the next box should appear

**Output:** A size-parameterized transducer  $C$  implementing the composition of  $\mathcal{T}$

```

1 Initialize  $\delta_C \leftarrow \emptyset$ ,  $Q_C \leftarrow \emptyset$ ,  $E_C \leftarrow \emptyset$ ;
2  $WList \leftarrow \{(\langle \text{root} \rangle, \text{bgn}, n)\}$ ;
3 while  $WList \neq \emptyset$  do
4   Pop ( $q = (\langle q_1, \dots, q_k \rangle, s)$ ,  $i$ ) from  $WList$ , add  $q$  to  $Q_C$  and, if  $\{q_1, \dots, q_k\} \subseteq E$ , also to  $E_C$ ;
5   foreach  $w \in \Gamma$  do
6      $(\ell, r) \leftarrow \delta(q_k, w) \circ \dots \circ \delta(q_2, w) \circ \delta(q_1, w)$ ;
7     if  $(\ell, r) = \perp$  then continue;
8     if  $i \neq 1 \vee s = \text{end}$  then // continuing in the current box(es)
9       if  $s = \text{bgn}$  then DiscoverTransition( $q \rightarrow w(\ell, r), s, i - 1$ );
10      else DiscoverTransition( $q \rightarrow w'(\ell, r), s, i - 1$ );
11    else // potentially start a new box
12      DiscoverTransition( $q \rightarrow w'(\ell, r), \text{end}, i - 1$ ); // no new box
13      if  $d = \text{left}$  then  $(\ell', r') \leftarrow (\ell, r) \circ (\text{root}(L), \text{root}(R))$ ; // new box to the left
14      else  $(\ell', r') \leftarrow (\text{root}(L), \text{root}(R)) \circ (\ell, r)$ ; // new box to the right
15      DiscoverTransition( $q \rightarrow w(\ell', r'), \text{bgn}, n$ );
16  return  $(Q_C, \delta_C, (\langle \text{root} \rangle, \text{bgn}), E_C)$ ;

17 Subroutine DiscoverTransition( $q \rightarrow u(\ell, r), \text{StateTag}, \text{Pos}$ ):
18    $\ell' \leftarrow \ell \{t \mapsto (t, \text{StateTag})\}$ ;  $r' \leftarrow r \{t \mapsto (t, \text{StateTag})\}$ ;
19    $\delta_C \leftarrow \delta_C \cup \{q \rightarrow u(\ell', r')\}$ ;
20   foreach  $p \in st(\ell') \cup st(r')$  do
21     if  $p \notin Q_C \wedge \nexists j(p, j) \in WList$  then Add  $(p, \text{Pos})$  to  $WList$ ;

```

rest of the input tree without spawning new MAJ instances). One could naturally implement this via nondeterminism, which is not possible with the WTT model since it is (top-down) deterministic. We can, however, get around the limitation by slightly changing the input alphabet: adding a primed version  $x'$  of the symbol  $x$ , which is used to label the last two qubits in the tree (encoding the fact that there is no further occurrence of MAJ). Naturally, one also needs to change the alphabet and structure of the input SWTA to make it compatible with the WTT, which may need its partial unfolding. In the case that we do not start a new instance of MAJ, we continue with the transition

$$\langle b \rangle \rightarrow x'(\langle f \rangle(L), \langle f \rangle(R)). \quad (28)$$

Alternatively, we can continue spawning more MAJ instances, using the transition

$$\langle b \rangle \rightarrow x(\langle f, a \rangle(L), \langle f, a \rangle(R)), \quad (29)$$

where the ' $a$ '-component of  $\langle f, a \rangle$  is the root state of MAJ, meaning that the next MAJ box is active.

Next, we compute the transition from  $\langle f, a \rangle$  similarly to transducer composition (cf. Section 5.8):

$$\langle f, a \rangle \rightarrow x(\langle l, b \rangle(L) + 0\langle l, c \rangle(R), \langle l, e \rangle(L) + 0\langle l, d \rangle(R)). \quad (30)$$

We, however, note that  $l$  is just an identity (the first transducer became inactive) and the state can therefore be removed from the tuples, yielding instead the transition

$$\langle f, a \rangle \rightarrow x(\langle b \rangle(L) + 0\langle c \rangle(R), \langle e \rangle(L) + 0\langle d \rangle(R)). \quad (31)$$

The full construction is more complex and is formalized in Algorithm 1. The input of the algorithm is a transducer  $\mathcal{T} = (Q, \delta, \text{root}, E)$  over  $\Gamma$  that needs to satisfy the following structural requirements:

- (1)  $Q$  contains the state  $\iota$  and  $E = \{\iota\}$ ,
- (2) for every  $a \in \Gamma \cup \Gamma'$ , the transition function  $\delta$  contains the transition  $\iota \rightarrow a(\iota(L), \iota(R))$ , and
- (3) all states from  $Q$  except  $\iota$  can occur only at a particular depth (distance from the root) in a run of  $\mathcal{T}$  on any tree (and the state  $\iota$  appears for the first time at the same depth on all branches); formally, there exists a function  $d: Q \rightarrow \mathbb{N}$  such that  $d(\text{root}) = 0$  and for every  $q \in Q \setminus \{\iota\}$ , if  $d(q) = m$  and  $q \rightarrow a(\ell, r) \in \delta$ , then for all  $p \in \text{st}(\ell) \cup \text{st}(r)$  we have that  $d(p) = m + 1$ .

Additional the inputs of the algorithm are the offset of boxes  $n$  ( $n = 2$  in the example above) and the side  $d \in \{\text{left}, \text{right}\}$  on which the boxes grow ( $d = \text{right}$  in the example).

The algorithm constructs a WTT with states of the form  $(\langle q_1, \dots, q_k \rangle, \text{Tag})$  where  $\langle q_1, \dots, q_k \rangle$  is a sequence of states of  $\mathcal{T}$  and  $\text{Tag} \in \{\text{bgn}, \text{end}\}$  denotes whether we can still spawn new boxes (bgn) or not any more (end). For a tuple  $\langle q_1, \dots, q_k \rangle$ , we assume all  $\iota$ 's are automatically deleted. The set  $WList$  keeps states to be explored, together with their offset (to know when to start spawning new instances of  $\mathcal{T}$ ). In the algorithm, we write  $\delta(q, a) = \perp$  if  $\delta$  is undefined on the input  $(q, a)$  and define the composition  $\circ$  of pairs of linear forms (used on Lines 6, 13, and 14) as follows: (i)  $\perp \circ x = x \circ \perp = \perp$  for any  $x$  and (ii)  $(\ell_2, r_2) \circ (\ell_1, r_1) = (\ell_2(\ell_1, r_1), r_2(\ell_1, r_1))$ , using the notation from Section 5.7. On Line 18, for a linear form  $v$ , we use  $v[\mathbf{t} \mapsto (\mathbf{t}, s)]$  to denote the linear form obtained from  $v$  by substituting every state (captured by)  $\mathbf{t}$  occurring in  $v$  by  $(\mathbf{t}, s)$ , e.g., for  $v = \langle b \rangle(L) + \langle c \rangle(R)$ , the result of  $v[\mathbf{t} \mapsto (\mathbf{t}, s)]$  is the linear form  $(\langle b \rangle, s)(L) + (\langle c \rangle, s)(R)$ .

## 7 Related Work

In recent years, many techniques for analyzing, simulating, and verifying quantum circuits and programs have emerged in the formal methods community. In this section, we provide their overview and explain where our approach stands among them.

*Symbolic Quantum Circuit Verifiers.* These tools are often fully automated and flexible in specifying different verification properties. The closest work to ours in this category are the automata-based approaches of [1, 4, 20–23], which use tree automata [27] and their variant *level-synchronized tree automata* (LSTAs) to encode predicates representing sets of quantum states. SWTAs generalize the previous tree automata models by the use of colors for synchronization across branches (this was already in LSTAs), weights, and possible interactions between multiple runs via linear forms. These techniques help SWTAs alleviate the scalability issues of the previous models in many cases where the representation using the previous model blew up. More concretely, our single-qubit gate operations are linear, while those of [20, 21] are exponential, and those in [4, 22] are quadratic (on the negative side, language inclusion/equivalence are undecidable now, but we use the decidable property of functional equivalence/inclusion to solve the verification problem). In addition, in the current paper, we propose a complete framework that uses transducers for implementing the quantum gates, whereas the previous works used specialized automata-manipulating algorithms.

There are a few more tools that belong to this category: *symQV* [8] is based on *symbolic execution* [38] to verify input-output relationship with queries discharged by SMT solvers over the theory of reals. SMT solvers have also been used in the verification of *quantum error correction code* (QECC) [16, 31]. The SMT array theory approach of [25] improved the previous by allowing a polynomial-sized circuit encoding, but still faces a similar scalability problem. Another scalable fully automated approach for analysis of quantum circuits is *quantum abstract interpretation* [49, 60], which, however, uses a quite coarse abstraction, which makes the properties that it can reason about quite limited. Among the approaches mentioned above, the one based on LSTAs [4, 22] is the only method that supports verification of quantum circuits where the size is a parameter. It can,

however, handle only a limited set of parametrized gates, e.g., it does not support the application of the Hadamard gate to every qubit, which is a common pattern in most interesting circuits.


*Deductive Quantum Circuit and Program Verifiers.* The tools in this category allow verification of quantum programs w.r.t. expressive specification languages. The most prominent family of approaches are those based on the so-called *quantum Hoare logic* [32, 41, 55, 59, 61]. These approaches, however, require significant manual work and user expertise (they are often based on the use of interactive theorem provers such as ISABELLE [48] and RocQ/CoQ [11]). The work in [36] adopts a syntactic approach that enables automatic reasoning within a fragment of quantum Hoare logic, and applies it to the verification of QECC. The work most closely related to ours in this category is QBRICKS [15], which also targets the verification of size-parametrized quantum circuits and leverages SMT solvers to discharge verification conditions automatically. While QBRICKS attempts to improve automation by generating proof obligations and solving them using SMT, the approach still requires substantial user interaction. For example, in the case of verifying Grover’s search for an arbitrary number of qubits, the experiment involved over one hundred manual interactions.

*Quantum Circuit Equivalence Checkers.* These tools are usually fully automated but are limited to only checking equivalence of two circuits. In contrast, our approach is flexible in specifying custom properties. Equivalence checkers are based on several approaches. One approach is based on the *ZX-calculus* [26], which is a graphical language used for reasoning about quantum circuits. The *path-sum* approach (implemented, e.g., within the tool FEYNMAN [7]) uses rewrite rules. Pre-computed equivalence sets are used to prove equivalence in QUARTZ [58]. QCEC [14] is an equivalence checker that uses decision diagrams and ZX-calculus, and SLIQC [19, 57] also uses decision diagrams for (partial) equivalence checking. An approach based on working with the so-called *stabilizer states* in [53] can be used to verify the equivalence of circuits with Clifford gates in polynomial time. There have also been works that approach quantum equivalence via model counting [44].

*Quantum Circuit Simulators.* Quantum circuit simulators can be broadly categorized into four classes: *decision-diagram-based* [17, 18, 45, 52, 54, 62], *state-vector-based* [40], *tensor-network-based* [42, 56], *ZX-calculus-based* [39], and *model-counting-based* [43]. These simulators are primarily designed to compute the output of a quantum circuit for a *single input state*. They can also be employed to verify circuit behavior over a finite number of input states by simulating each one individually. However, they cannot be used for size-parametrized verification, which can be handled by the SWTA-based framework.

SWTAs can be viewed as a generalization of decision diagrams, particularly the *quantum multiple-valued decision diagrams* (QMDDs) [62] and *tensor decision diagrams* [35], which place weights on edges to compactly represent quantum states. Unlike traditional simulators, our method is designed to handle *sets of input states simultaneously*, improving scalability.

## Acknowledgments

We thank the anonymous reviewers for their feedback that improved the quality of the paper. This work was supported by the Czech Science Foundation projects 25-18318S and 25-17934S; the FIT BUT internal project FIT-S-23-8151; National Science and Technology Council, R.O.C., projects NSTC 114-2221-E-027-044 -MY2 and NSTC 114-2119-M-001-002-; Air Force Office of Scientific Research project FA2386-23-1-4107; and Academia Sinica Investigator Project Grant AS-IV-114-M07. The work of Michal Hečko, a Brno Ph.D. Talent Scholarship  Holder, is funded by the Brno City Municipality.

## References

- [1] Parosh Aziz Abdulla. 2025. A Symbolic Approach to Verifying Quantum Systems. *Commun. ACM* 68, 6 (June 2025), 84. doi:10.1145/3725725
- [2] Parosh Aziz Abdulla, Yu-Fang Chen, Michal Hečko, Lukáš Holík, Ondřej Lengál, Jyun-Ao Lin, and Ramanathan Srinivasan Thinniyam. 2025. Parameterized Verification of Quantum Circuits (Technical Report). *CoRR* 2511.19897 (2025). arXiv:2511.19897 <https://arxiv.org/abs/2511.19897>
- [3] Parosh Aziz Abdulla, Yu-Fang Chen, Michal Hečko, Lukáš Holík, Ondřej Lengál, Jyun-Ao Lin, and Ramanathan S. Thinniyam. 2025. AUTOQ-PARA repository. <https://github.com/VeriFIT/AutoQ-Para>
- [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. *Proceedings of the ACM on Programming Languages* 9, POPL (2025), 923–953.
- [5] 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
- [6] Rajeev Alur and David L. Dill. 1994. A Theory of Timed Automata. *Theor. Comput. Sci.* 126, 2 (1994), 183–235. doi:10.1016/0304-3975(94)90010-8
- [7] 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
- [8] Fabian Bauer-Marquart, Stefan Leue, and Christian Schilling. 2023. symQV: Automated Symbolic Verification of Quantum Programs. In *Formal Methods - 25th International Symposium, FM 2023, Lübeck, Germany, March 6-10, 2023, Proceedings (Lecture Notes in Computer Science, Vol. 14000)*, Marsha Chechik, Joost-Pieter Katoen, and Martin Leucker (Eds.). Springer, 181–198. doi:10.1007/978-3-031-27481-7\_12
- [9] Rodney J Baxter. 1972. One-dimensional anisotropic Heisenberg chain. *Annals of Physics* 70, 2 (1972), 323–337. doi:10.1016/0003-4916(72)90270-9
- [10] Ethan Bernstein and Umesh V. Vazirani. 1993. Quantum complexity theory. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal (Eds.). ACM, 11–20. doi:10.1145/167088.167097
- [11] Yves Bertot and Pierre Castéran. 2013. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media.
- [12] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. 2002. Quantum amplitude amplification and estimation. In *Quantum computation and information (Washington, DC, 2000)*. Contemp. Math., Vol. 305. Amer. Math. Soc., Providence, RI, 53–74. doi:10.1090/conm/305/05215
- [13] Gilles Brassard, Peter Høyer, and Alain Tapp. 1998. Quantum Counting. In *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings (Lecture Notes in Computer Science, Vol. 1443)*, Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel (Eds.). Springer, 820–831. doi:10.1007/BFB0055105
- [14] Lukas Burgholzer and Robert Wille. 2020. Advanced equivalence checking for quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 9 (2020), 1810–1824. doi:10.1109/TCAD.2020.3032630
- [15] Christophe Charetton, Sébastien Bardin, François Bobot, Valentin Perrelle, and Benoît Valiron. 2021. An Automated Deductive Verification Framework for Circuit-Building Quantum Programs. In *ESOP (LNCS, Vol. 12648)*, Nobuko Yoshida (Ed.). Springer International Publishing, Cham, 148–177. doi:10.1007/978-3-030-72019-3\_6
- [16] Kean Chen, Yuhao Liu, Wang Fang, Jennifer Paykin, Xin-Chuan Wu, Albert Schmitz, Steve Zdancewic, and Gushu Li. 2025. Verifying Fault-Tolerance of Quantum Error Correction Codes. In *Computer Aided Verification, 12th International Conference, CAV 2025 (LNCS)*. Springer.
- [17] Tian-Fu Chen, Yu-Fang Chen, Jie-Hong Roland Jiang, Sára Jobranová, and Ondřej Lengál. 2024. Accelerating Quantum Circuit Simulation with Symbolic Execution and Loop Summarization. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2024, Newark Liberty International Airport Marriott, NJ, USA, October 27-31, 2024*, Jinjun Xiong and Robert Wille (Eds.). ACM, 42:1–42:9. doi:10.1145/3676536.3676711
- [18] Tian-Fu Chen and Jie-Hong R. Jiang. 2025. SliQSim: A Quantum Circuit Simulator and Solver for Probability and Statistics Queries. In *Tools and Algorithms for the Construction and Analysis of Systems - 31st International Conference, TACAS 2025, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, ON, Canada, May 3-8, 2025, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 15698)*, Arie Gurfinkel and Marijn Heule (Eds.). Springer, 129–138. doi:10.1007/978-3-031-90660-2\_7
- [19] Tian-Fu Chen, Jie-Hong R. Jiang, and Min-Hsiu Hsieh. 2022. Partial Equivalence Checking of Quantum Circuits. In *IEEE International Conference on Quantum Computing and Engineering, QCE 2022, Broomfield, CO, USA, September 18-23, 2022*. IEEE, 594–604. doi:10.1109/QCE53715.2022.00082

- [20] Yu-Fang Chen, Kai-Min Chung, Ondřej 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
- [21] Yu-Fang Chen, Kai-Min Chung, Ondřej 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
- [22] 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 *TACAS 2025*. Springer, 87–108.
- [23] 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
- [24] Yu-Fang Chen, Ondřej Lengál, Tony Tan, and Zhilin Wu. 2017. Register automata with linear arithmetic. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 1–12. doi:10.1109/LICS.2017.8005111
- [25] Yu-Fang Chen, Philipp Rümmer, and Wei-Lun Tsai. 2023. A Theory of Cartesian Arrays (with Applications in Quantum Circuit Verification). In *International Conference on Automated Deduction*. Springer, 170–189.
- [26] Bob Coecke and Ross Duncan. 2011. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics* 13, 4 (apr 2011), 043016. doi:10.1088/1367-2630/13/4/043016
- [27] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. 2008. Tree automata techniques and applications.
- [28] Steven A Cuccaro, Thomas G Draper, Samuel A Kutin, and David Petrie Moulton. 2004. A new quantum ripple-carry addition circuit. *arXiv preprint quant-ph/0410184* (2004).
- [29] Manfred Droste, Werner Kuich, and Heiko Vogler. 2009. *Handbook of Weighted Automata*. Springer.
- [30] Javier Esparza and Michael Blondin. 2023. *Automata Theory: An Algorithmic Approach*. MIT Press.
- [31] Wang Fang and Mingsheng Ying. 2024. Symbolic execution for quantum error correction programs. *Proceedings of the ACM on Programming Languages* 8, PLDI (2024), 1040–1065.
- [32] 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
- [33] Daniel M. Greenberger, Michael A. Horne, and Anton Zeilinger. 1989. Going Beyond Bell’s Theorem. In *Bell’s Theorem, Quantum Theory and Conceptions of the Universe*, Menas Kafatos (Ed.). Springer Netherlands, Dordrecht, 69–72. doi:10.1007/978-94-017-0849-4\_10
- [34] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, Gary L. Miller (Ed.). ACM, 212–219. doi:10.1145/237814.237866
- [35] Xin Hong, Xiangzhen Zhou, Sanjiang Li, Yuan Feng, and Mingsheng Ying. 2022. A Tensor Network based Decision Diagram for Representation of Quantum Circuits. *ACM Trans. Design Autom. Electr. Syst.* 27, 6 (2022), 60:1–60:30. doi:10.1145/3514355
- [36] Qifan Huang, Li Zhou, Wang Fang, Mengyu Zhao, and Mingsheng Ying. 2025. Efficient Formal Verification of Quantum Error Correcting Programs. *Proc. ACM Program. Lang.* 9, PLDI, Article 190 (June 2025), 26 pages. doi:10.1145/3729293
- [37] Michael Karr. 1976. Affine Relationships Among Variables of a Program. *Acta Informatica* 6 (1976), 133–151. doi:10.1007/BF00268497
- [38] James C. King. 1976. Symbolic Execution and Program Testing. *Commun. ACM* 19, 7 (1976), 385–394. doi:10.1145/360248.360252
- [39] Aleks Kissinger and John van de Wetering. 2022. Simulating quantum circuits with ZX-calculus reduced stabiliser decompositions. *Quantum Science and Technology* 7, 4 (jul 2022), 044001. doi:10.1088/2058-9565/ac5d20
- [40] Ang Li and Sriram Krishnamoorthy. 2021. SV-Sim: Scalable PGAS-based State Vector Simulation of Quantum Circuits. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.
- [41] Junyi Liu, Bohua Zhan, Shuling Wang, Shenggang Ying, Tao Liu, Yangjia Li, Mingsheng Ying, and Naijun Zhan. 2019. Formal verification of quantum algorithms using quantum Hoare logic. In *International conference on computer aided verification*. Springer, 187–207. doi:10.1007/978-3-030-25543-5\_12
- [42] Igor L Markov and Yaoyun Shi. 2008. Simulating quantum computation by contracting tensor networks. *SIAM J. Comput.* 38, 3 (2008), 963–981.
- [43] Jingyi Mei, Marcello M. Bonsangue, and Alfons Laarman. 2024. Simulating Quantum Circuits by Model Counting. In *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 14683)*, Arie Gurfinkel and Vijay Ganesh (Eds.). Springer, 555–578. doi:10.1007/978-3-031-65633-0\_25

- [44] Jingyi Mei, Tim Coopmans, Marcello M. Bonsangue, and Alfons Laarman. 2024. Equivalence Checking of Quantum Circuits by Model Counting. In *Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, July 3-6, 2024, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 14740)*, Christoph Benzmüller, Marijn J. H. Heule, and Renate A. Schmidt (Eds.). Springer, 401–421. doi:10.1007/978-3-031-63501-4\_21
- [45] D. Michael Miller and Mitchell A. Thornton. 2006. QMDD: A Decision Diagram Structure for Reversible and Quantum Circuits. In *36th IEEE International Symposium on Multiple-Valued Logic (ISMVL 2006), 17-20 May 2006, Singapore*. IEEE Computer Society, 30. doi:10.1109/ISMVL.2006.35
- [46] Markus Müller-Olm and Helmut Seidl. 2004. A note on Karr’s algorithm. In *International Colloquium on Automata, Languages, and Programming*. Springer, 1016–1028.
- [47] Michael A. Nielsen and Isaac L. Chuang. 2011. *Quantum Computation and Quantum Information: 10th Anniversary Edition* (10th ed.). Cambridge University Press, USA.
- [48] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. 2002. *Isabelle/HOL: a proof assistant for higher-order logic*. Vol. 2283. Springer Science & Business Media.
- [49] Simon Perdrix. 2008. Quantum entanglement analysis based on abstract interpretation. In *International Static Analysis Symposium*. Springer, 270–282. doi:10.1007/978-3-540-69166-2\_18
- [50] Asher Peres. 1985. Reversible logic and quantum computers. *Phys. Rev. A* 32 (Dec 1985), 3266–3276. Issue 6. doi:10.1103/PhysRevA.32.3266
- [51] Emil Post. 1946. A variant of a recursively unsolvable problem. *Bull. Amer. Math. Soc.* 52 (1946). doi:10.1090/S0002-9904-1946-08555-9
- [52] Meghana Sistla, Swarat Chaudhuri, and Thomas W. Reps. 2023. Symbolic Quantum Simulation with Quasimodo. 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, 213–225. doi:10.1007/978-3-031-37709-9\_11
- [53] Dimitrios Thanos, Tim Coopmans, and Alfons Laarman. 2023. Fast Equivalence Checking of Quantum Circuits of Clifford Gates. In *Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 14216)*, Étienne André and Jun Sun (Eds.). Springer, 199–216. doi:10.1007/978-3-031-45332-8\_10
- [54] Yuan-Hung Tsai, Jie-Hong R. Jiang, and Chiao-Shan Jhang. 2021. Bit-Slicing the Hilbert Space: Scaling Up Accurate Quantum Circuit Simulation. In *58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021*. IEEE, 439–444. doi:10.1109/DAC18074.2021.9586191
- [55] 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
- [56] Guifré Vidal. 2003. Efficient Classical Simulation of Slightly Entangled Quantum Computations. *Phys. Rev. Lett.* 91 (Oct 2003), 147902. Issue 14. doi:10.1103/PhysRevLett.91.147902
- [57] Chun-Yu Wei, Yuan-Hung Tsai, Chiao-Shan Jhang, and Jie-Hong R. Jiang. 2022. Accurate BDD-based unitary operator manipulation for scalable and robust quantum circuit verification. In *DAC ’22: 59th ACM/IEEE Design Automation Conference, San Francisco, California, USA, July 10 - 14, 2022*, Rob Oshana (Ed.). ACM, 523–528. doi:10.1145/3489517.3530481
- [58] Mingkuan Xu, Zikun Li, Oded Padon, Sina Lin, Jessica Pointing, Auguste Hirth, Henry Ma, Jens Palsberg, Alex Aiken, Umut A Acar, et al. 2022. Quartz: superoptimization of Quantum circuits. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 625–640. doi:10.1145/3519939.3523433
- [59] 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
- [60] Nengkun Yu and Jens Palsberg. 2021. Quantum abstract interpretation. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 542–558. doi:10.1145/3453483.3454061
- [61] 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
- [62] Alwin Zulehner and Robert Wille. 2019. Advanced Simulation of Quantum Computations. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 38, 5 (2019), 848–859. doi:10.1109/TCAD.2018.2834427

Received 2025-07-10; accepted 2025-11-06