Synthesizing Quantum-Circuit Optimizers

AMANDA XU, University of Wisconsin-Madison, USA
ABTIN MOLAVI, University of Wisconsin-Madison, USA
LAUREN PICK, University of Wisconsin-Madison, USA
SWAMIT TANNU, University of Wisconsin-Madison, USA
AWS ALBARGHOUTHI, University of Wisconsin-Madison, USA

Near-term quantum computers are expected to work in an environment where each operation is noisy, with no error correction. Therefore, quantum-circuit optimizers are applied to minimize the number of noisy operations. Today, physicists are constantly experimenting with novel devices and architectures. For every new physical substrate and for every modification of a quantum computer, we need to modify or rewrite major pieces of the optimizer to run successful experiments. In this paper, we present QUESO, an efficient approach for automatically synthesizing a quantum-circuit optimizer for a given quantum device. For instance, in 1.2 minutes, QUESO can synthesize an optimizer with high-probability correctness guarantees for IBM computers that significantly outperforms leading compilers, such as IBM's Qiskit and TKET, on the majority (85%) of the circuits in a diverse benchmark suite.

A number of theoretical and algorithmic insights underlie QUESO: (1) An algebraic approach for representing rewrite rules and their semantics. This facilitates reasoning about complex *symbolic* rewrite rules that are beyond the scope of existing techniques. (2) A fast approach for probabilistically verifying equivalence of quantum circuits by reducing the problem to a special form of *polynomial identity testing*. (3) A novel probabilistic data structure, called a *polynomial identity filter* (PIF), for efficiently synthesizing rewrite rules. (4) A beam-search-based algorithm that efficiently applies the synthesized symbolic rewrite rules to optimize quantum circuits.

CCS Concepts: • **Software and its engineering** → **Compilers**; • **Hardware** → **Quantum computation**.

Additional Key Words and Phrases: quantum computing, probabilistic verification

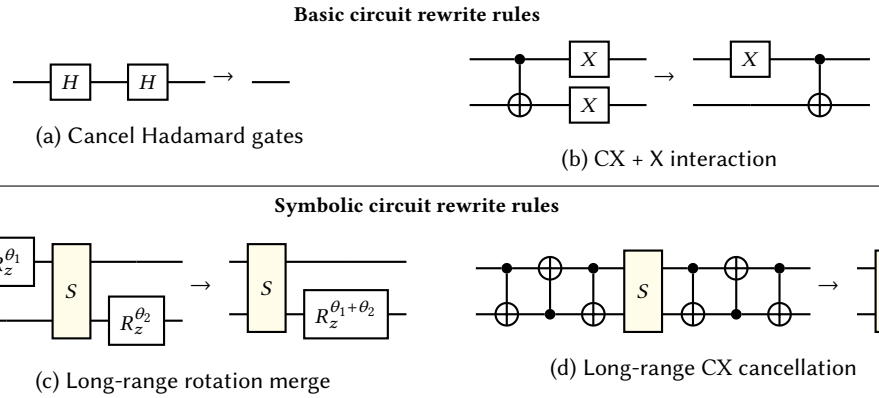
ACM Reference Format:
Amanda Xu, Abtin Molavi, Lauren Pick, Swamit Tannu, and Aws Albarghouthi. 2023. Synthesizing Quantum-Circuit Optimizers. *Proc. ACM Program. Lang.* 7, PLDI, Article 140 (June 2023), 25 pages. https://doi.org/10.1145/3591254

1 INTRODUCTION

The dream of quantum computing has been around for decades, but it is only recently that we have begun to witness promising physical realizations of quantum computers. Quantum computers enable efficient simulation of quantum mechanical phenomena, potentially opening the door to advances in quantum physics, chemistry, material design, and beyond. Near-term quantum computers with several dozens of qubits are expected to operate in a noisy environment without error correction, in a model of computation called *Noisy Intermediate Scale Quantum* (NISQ) computing [Preskill 2018].

Authors' addresses: Amanda Xu, University of Wisconsin-Madison, Madison, WI, USA, axu44@wisc.edu; Abtin Molavi, University of Wisconsin-Madison, Madison, WI, USA, amolavi@wisc.edu; Lauren Pick, University of Wisconsin-Madison, Madison, WI, USA, lpick2@wisc.edu; Swamit Tannu, University of Wisconsin-Madison, Madison, WI, USA, stannu@wisc.edu; Aws Albarghouthi, University of Wisconsin-Madison, Madison, WI, USA, aws@cs.wisc.edu.

2475-1421/2023/6-ART140
https://doi.org/10.1145/3591254



(a) Cancel Hadamard gates

(b) CX + X interaction

(c) Long-range rotation merge

(d) Long-range CX cancellation

Fig. 1. Some optimizations QUESO can synthesize/verify (S is a symbolic gate satisfying some constraints)

In NISQ computers, each operation is noisy. Therefore, powerful quantum-circuit optimizers are absolutely crucial: we want to produce smaller circuits that are more tolerant to noise. Without careful optimization, one can easily end up with a circuit whose results are indistinguishable from random noise. However, the state of quantum hardware is in flux. There are so many physical realizations of quantum computers, and physicists are constantly experimenting with new devices and architectures—*neutral atoms*, *superconducting circuits*, *semiconductor devices* [Saffman 2019; Watson et al. 2018; Wilen et al. 2021]. **For every new physical substrate and for every modification of a quantum computer, we need to modify or rewrite major pieces of the optimizer to run experiments.** This is a bottleneck in our progress towards a quantum computing future: writing optimizers is a tedious, iterative, heuristic process, and one that is error-prone [Paltenghi and Pradel 2022].

Our goal in this paper is to answer the following question:

Given a specification of a quantum architecture, can we automatically synthesize an efficient and correct quantum-circuit optimizer?

Recent developments only partially address this question: The quantum-circuit optimizer, VOQC [Hietala et al. 2021], is manually written with machine-checked correctness proofs, and therefore is not automatically extensible to new quantum architectures. The superoptimizer, Quartz [Xu et al. 2022a], automatically synthesizes semantics-preserving circuit rewrite rules; however, it can only synthesize simple rewrite rules and, as a superoptimizer, is heavily dependent on hand-crafted, device-specific optimization passes without which the synthesized rules have little impact.

We present QUESO, a new technique that rapidly synthesizes sophisticated, correct rewrite rules. QUESO then efficiently applies the synthesized rules to optimize quantum circuits. QUESO builds upon four critical ideas: (1) An algebraic approach for representing rewrite rules and their semantics. This allows us to reason about and synthesize complex optimizations beyond the scope of existing techniques. (2) A fast probabilistic verification approach for checking rewrite-rule correctness by reducing the problem to a special form of *polynomial identity testing* and demonstrating that the standard fast randomized algorithm applies. (3) A probabilistic data structure for efficiently synthesizing equivalent pairs of circuits without incurring a quadratic explosion. (4) A beam-search-based algorithm that efficiently applies synthesized rewrite rules to optimize circuits.

Symbolic rules. Typically, quantum-circuit optimizers apply a schedule of rewrite rules to shrink a given circuit. In its simplest form, a rewrite rule matches a specific subcircuit and rewrites it into a smaller, equivalent subcircuit. For instance, Fig. 1a shows two equivalent circuits: if we see two

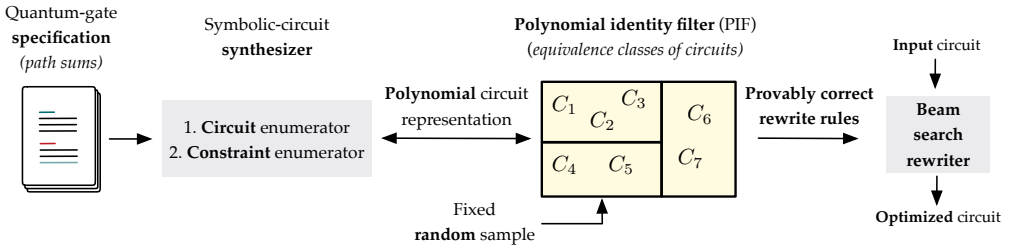


Fig. 2. Overview of the QUESO approach

Hadamard gates (H) applied to the same qubit, we can eliminate them because they cancel each other out. Fig. 1b shows a rewrite rule over subcircuits with two qubits.

Automatically synthesizing such rules is relatively simple: enumerate pairs of circuits and verify their equivalence. This approach that has been applied in other domains, e.g., machine learning [Jia et al. 2019], traditional compilers [Sasnauskas et al. 2017], and recently quantum-circuit superoptimization [Xu et al. 2022a]. However, there are complex and critical rules that cannot be discovered this way: subcircuits can have parameters, e.g., angles of rotations, or *completely unknown subcircuits*. Thus, we need a *symbolic approach* for reasoning about such rules. For instance, Fig. 1c shows a rewrite rule in which two rotations about the z-axis on different qubits can be merged into a single rotation, even if they are separated by arbitrarily many operations, denoted S, so long as S satisfies certain conditions. We think of S as an unknown, *symbolic* gate. Similarly, Fig. 1d shows a rule in which two distant sequences of CX gates can be cancelled.

To reason about symbolic circuits and rules, we utilize *path-sum*-based semantics. First introduced by Feynman, path sums compactly capture the semantics of a quantum system as an expression. Intuitively, one can think of a path sum as a transition relation specifying how a quantum system's state evolves. Indeed, path sums have been used for quantum-circuit verification [Amy 2019; Chareton et al. 2021]. In this paper, we exploit the algebraic nature of path sums to reason about circuits with unknown parameters and unknown subcircuits. By reasoning using path sums, we show that we are able to synthesize *long-range* rules, like Figs. 1c and 1d, that cancel out far-apart quantum gates.

Probabilistic verification and synthesis. QUESO synthesizes rewrite rules following the standard synthesize-and-verify story: we enumerate circuits with symbolic components and verify equivalence of pairs of circuits. If two circuits C_1 and C_2 are proven equivalent, then we can soundly rewrite C_1 to C_2 or vice versa. Naïvely following this recipe, of course, does not scale due to the large space of pairs of circuits. Our first, and perhaps most critical, observation is that the equivalence-checking problem for two circuits can be reduced to a *constrained* form of *polynomial identity testing* (PIT)—the problem of checking equivalence of two polynomials—where the constraints are on the domain of the variables. We then demonstrate that this problem can be directly solved by the foundational Schwartz–Zippel randomized algorithm for PIT [Motwani and Raghavan 1995, Ch. 7], which is very fast, because it relies on a single random instantiation of the variables of a polynomial.

With this insight, we present a probabilistic data structure—the *polynomial identity filter* (PIF)—for constructing equivalence classes of circuits. The PIF builds upon the high-probability guarantees of Schwartz–Zippel to eliminate the quadratic explosion of checking equivalence of pairs of circuits. The PIF therefore enables fast construction of rewrite rules from equivalence classes.

Applying rewrite rules. Quantum-circuit optimizers, like VOQC [Hietala et al. 2021] and TKET [Sivarajah et al. 2020], use a fixed schedule for applying optimizations that is chosen by

the compiler developer. In our setting, however, we synthesize tens of thousands of rules, and we simply cannot ask a developer to experiment with different schedules. We demonstrate that a simple, beam-search-based algorithm can quickly optimize quantum circuits by applying sequences of rewrite rules. The most algorithmically challenging piece is matching symbolic rewrite rules, which can match arbitrarily large subcircuits.

Evaluation. We implemented QUESO and used it to synthesize optimizers for four different quantum architectures with different operations (or gate sets)—including IBM, Rigetti, and *ion trap* computers. QUESO can synthesize all rewrite rules in about 2 minutes. Our results demonstrate that QUESO is able to outperform or match handwritten optimizers, like VOQC [Hietala et al. 2021], TKET [Sivarajah et al. 2020], and IBM Qiskit [Aleksandrowicz et al. 2019]. For instance, in 1.2 minutes, QUESO can synthesize an optimizer for IBM computers that significantly outperforms leading compilers, such as IBM's Qiskit and TKET, on the majority (85%) of the circuits in a diverse benchmark suite, and can outperform or match VOQC on 72% of the benchmarks, outperforming it in 51% of the benchmarks. In comparison to the superoptimizer, Quartz [Xu et al. 2022a], we demonstrate (1) that QUESO is radically faster at rule synthesis and (2) the critical importance of symbolic rewrite rules. For instance, on IBM, in a head-to-head comparison of synthesized rules (i.e., excluding any preprocessing), QUESO synthesizes rules an order of magnitude faster than Quartz and outperforms Quartz on 97% of the benchmarks.

Contributions. In summary, we make the following contributions:

- A path-sum-based circuit semantics that can compactly capture circuits with unknown, symbolic subcircuits. This enables us to synthesize sophisticated, long-range rewrite rules that are critical for quantum-circuit optimization. (§ 3)
- A fast quantum-circuit equivalence verifier that reduces the problem to a constrained form of polynomial identity testing, which can be solved using a fast randomized algorithm. (§ 4)
- A fast rule synthesizer that uses a novel probabilistic data structure, called *polynomial identity filter* (PIF), to avoid the quadratic explosion of rule enumeration. (§ 5)
- A beam-search-like algorithm that applies symbolic rewrite rules to optimize a circuit. (§ 6)
- A thorough evaluation of QUESO on four quantum architectures. Our results demonstrate that QUESO can outperform or match state-of-the-art optimizers. (§ 7)

The full version of this paper is [Xu et al. 2022b].

2 BACKGROUND AND OVERVIEW

2.1 Quantum Circuits Background

Quantum state. A quantum bit (qubit) can be in state 0 or 1, the *computational basis states*, which are represented by the 2-dimensional vectors $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, respectively. A qubit can also be in a linear combination (*superposition*) of the basis states, $\alpha\,|0\rangle + \beta\,|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$, where α, β are complex numbers, called the *amplitudes*, such that $|\alpha|^2 + |\beta|^2 = 1$. The state of two qubits is a vector of four complex numbers, where each number is the amplitude of one of the basis states, $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$. The state of n qubits is a vector of 2^n complex numbers.

Quantum gates. Quantum operations (or *gates*) transform the state of the qubits of a system. Unlike in Boolean circuits, there are infinitely many possible quantum gate combinations that can be used to produce a *universal* quantum computer—one that can approximate arbitrary *unitary* transformations (the class of state transformations the rules of quantum mechanics admit) to arbitrary precision. Because of a variety of engineering challenges, different quantum computers provide different gate sets. We give examples of some standard gates below.

A classical gate like NOT (denoted X) can be applied to a single qubit. If the qubit state is $|0\rangle$, it becomes $|1\rangle$, and vice versa, just like on a classical circuit. However, if the state of the qubit is a superposition $\alpha\,|0\rangle + \beta\,|1\rangle$, applying X results in the state $\beta\,|0\rangle + \alpha\,|1\rangle$, i.e., swaps the amplitudes. The *Hadamard* gate, denoted H, takes a qubit from a basis state and puts it in superposition; for example, given the basis state $|0\rangle$, applying H results in $\frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle$.

Path sums. Since quantum operations are *linear* transformations, they are represented uniquely by how they transform the basis states. We use the traditional *path-sum* notation [Amy 2019], which can be seen as a compact representation of a state-transition relation. For example, the path-sum representation of the X, H, and R_z gates are defined as follows:

$$X : |x\rangle \to |\neg x\rangle \qquad H : |x\rangle \to \frac{1}{\sqrt{2}} \sum_{y \in \{0,1\}} e^{i\pi xy}\,|y\rangle \qquad R_z^\theta : |x\rangle \to e^{i(2x-1)\theta}\,|x\rangle$$

These are read as follows: Applying gate X to basis state $|x\rangle$ results in the state $|\neg x\rangle$; applying H to $|x\rangle$ results in the state $\frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}} e^{i\pi x}\,|1\rangle$. The R_z gate is parameterized by an angle θ, and only changes the amplitude of a given basis state. The *controlled X* (CX) gate can *entangle* two qubits, a critical operation in quantum computing: $|x_1x_2\rangle \to |x_1(x_1 \oplus x_2)\rangle$. Given a basis state $|x_1x_2\rangle$, CX produces the basis state $|x_1(x_1 \oplus x_2)\rangle$, where $\oplus$ is XOR.

Quantum circuits. Quantum circuits are combinations of quantum gates. Consider the circuit in Fig. 3 (left) over two qubits, x_1 and x_2, represented by the two horizontal *wires*. The circuit is read from left to right. The first gate is a CX gate applied to the two qubits. Then, an X gate and an $R_z^{\pi/2}$ gate are applied to x_1 and x_2 in parallel.

We will use a linear representation of the circuit as a sequence of gates—Fig. 3 (right). Since X and R_z are applied in parallel to two different qubits, we can safely swap them in the linear representation. The semantics of the circuit can be represented in path-sum notation by *composing* the path-sum representation of the constituent gates.

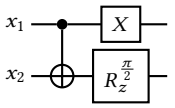


Fig. 3. Circuit and its linear representation

Executing circuits on hardware. On quantum hardware, operations are imperfect and prone to errors. Typically, quantum computers provide single- and two-qubit gates, which are optimized to minimize errors. Despite these optimizations, the average single-qubit gate error rate is about 0.1%, *while the two-qubit gate error is about 10-100x higher* on most industrial quantum computers [IBM 2022; Quantum-AI 2021]. Therefore, to minimize circuit error rate, we need to eliminate as many two-qubit gates as possible, which not only corrupt the qubits involved in the operation, but also impose *crosstalk* errors on neighboring qubit devices, significantly degrading circuit reliability. See full version of this paper for more details on hardware and errors.

2.2 Overview of QUESO

High-level view. Fig. 2 provides a high-level illustration of QUESO. First, the user provides a specification of the gate set of some quantum architecture, e.g., IBM's gate set, in the form of path sums. Then, a synthesis algorithm starts enumerating (symbolic) circuits along with constraints on the symbolic components. All circuits are inserted into a probabilistic data structure—the polynomial identity filter—which groups circuits into correct equivalence classes with high-probability guarantees. Finally, we construct rewrite rules from equivalent pairs of circuits and apply them following a beam-search-based algorithm to optimize a given circuit. We illustrate these pieces with examples.

Example A. Let us consider the rewrite rule in Fig. 1c. The two circuits on either side of the rule have three unknowns: the rotation angles, θ_1 and θ_2, and the highlighted gate S. Our synthesis algorithm enumerates such circuits in order to discover equivalent pairs of circuits. However, in our example, the gate S is completely unconstrained—we know nothing about it. We call S a *symbolic* gate. Therefore, QUESO needs to answer the following *abduction* question:

Under what conditions on S are the two circuits equivalent?

For this specific example, QUESO abduces the following constraint on S:

$$S : |x_1 x_2 \ldots\rangle \to \phi(x_1 x_2 \ldots)\,|x_2 x_1 \ldots\rangle$$

In other words, all we need to know about S for these two circuits to be equivalent is that S swaps the values of x_1 and x_2. Note that S is allowed to change the amplitudes (denoted by an unconstrained function ϕ, called the *amplitude transformer*) and may even apply gates to other qubits (denoted by the $\ldots$).

Example B. Fig. 1d shows another rewrite rule that QUESO can synthesize. This rewrite rule cancels two distant sequences of CX gates, separated by a symbolic gate S. QUESO abduces the following constraint on S.

$$S : |x_1 x_2 \ldots\rangle \to \phi(x_1 x_2 \ldots)\,|x_1 x_2 \ldots\rangle$$

Informally, S may change the amplitudes but should not change the first two bits of the state, $x_1 x_2$.

Proving equivalence. To prove equivalence of two circuits, we observe that we can reduce the problem to a constrained form of *polynomial identity testing* (PIT), the problem of checking equivalence of two polynomials. Specifically, the amplitudes of every basis state will be represented as a polynomial over the complex field. In Example A, the polynomials describing the amplitudes will be over the variables θ_1, θ_2, and the function ϕ (from the constraint on S).

We show that our constrained PIT problem can be solved with the standard randomized algorithm—following the Schwartz–Zippel lemma [Motwani and Raghavan 1995]. Simply, randomly sample values for the variables and check if the two polynomials evaluate to the same value. If they do not, then we have a counterexample; if they do, then we have a high-probability guarantee that the polynomials are equivalent.

To give some intuition, for Example A (Fig. 1c), the amplitudes of the basis state $|11\rangle$ for the left and right circuits are as follows (a full derivation is shown in Example 3.9):

$$e^{i\theta_1} \cdot \phi(11) \cdot e^{i\theta_2} = \phi(11) \cdot e^{i(\theta_1+\theta_2)}$$

where ϕ is the unknown amplitude transformer of S. While these two expressions are clearly equivalent, it is not always immediate, and one generally requires algebraic manipulation to prove equivalence. Further, these two expressions are not polynomials. Luckily, as we show in § 4, we demonstrate that we can treat the above expressions as a special form of polynomials over the complex field and use Schwartz–Zippel to show their equivalence. Specifically, $\phi(11)$ can be viewed as a complex variable and terms of the form $e^{i\cdot}$ as complex variables *constrained* to the unit circle.

Efficient synthesis. To synthesize rewrite rules, we can simply enumerate pairs of circuits, abduce constraints, and verify their equivalence. But this blows up quadratically—say there are 1 million circuits, then we will need to consider 10^{12} pairs.

To avoid the quadratic explosion we present a simple and efficient probabilistic data structure, the *polynomial identity filter* (PIF). The PIF takes a set of polynomials and returns a set of equivalence classes. The key idea underlying the PIF is that we can use a single random valuation of variables to evaluate each circuit's polynomial representation, and store circuits with equal valuations in the same equivalence class. Using the guarantees of Schwartz–Zippel, the PIF data structure ensures

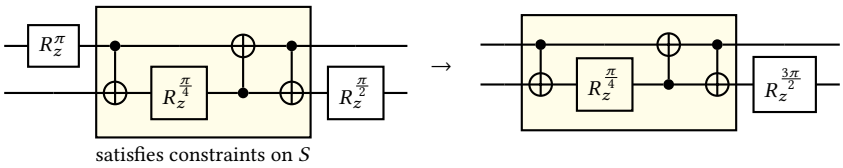


Fig. 4. Application of the long-range rotation merge optimization from Fig. 1c.

that all of its equivalence classes are correct with a high probability (Thm. 5.1). From the generated equivalence classes, we construct a set of rewrite rules like those shown in Fig. 1.

Applying rules. Given a set of rewrite rules, QUESO uses a beam search approach to traverse the space of rewrite sequences and optimize the circuit by minimizing the number of gates. The main challenge that QUESO addresses is a generic algorithm for applying symbolic rewrite rules. Specifically, it needs to discover a subcircuit that satisfies the constraints on the symbolic gate S in our examples. Fig. 4 demonstrates an application of the long-range rotation merge rule (Fig. 1c) to a circuit. The highlighted part of the circuit satisfies the constraints on the symbolic gate S, namely, the path-sum representation of the highlighted subcircuit is of the form $|x_1x_2\ldots\rangle \to \phi(x_1x_2)\,|x_2x_1\ldots\rangle$.

3 PATH-SUM-BASED CIRCUIT SEMANTICS

We now formally define (symbolic) quantum circuits and their semantics using *path sums*. Our semantics is a direct adaptation of that of Amy [2019]. The novelty in this section is defining circuits with unknown, symbolic gates and what it means for such circuits to be equivalent (§ 3.3).

3.1 States, Gates, and Path Sums

Quantum states. The state of a qubit is a linear combination of the computational basis states, $|0\rangle$ and $|1\rangle$, written $\alpha\,|0\rangle + \beta\,|1\rangle$, where $\alpha, \beta \in \mathbb{C}$. The state of n qubits is a term of the form $\sum_{\mathbf{x}\in\mathbb{Z}_2^n} \alpha_{\mathbf{x}}\,|\mathbf{x}\rangle$, where $\mathbb{Z}_2^n$ is the set of n-bit vectors and $\alpha_{\mathbf{x}} \in \mathbb{C}$. For bit vector $\mathbf{x}$, we use x_i to denote the ith bit of $\mathbf{x}$.

Gates and path sums. We consider two kinds of quantum gates, single- and multi-qubit gates. We will use G^{ρ} to denote a gate that takes a parameter $\rho \in \mathbb{C}$ (e.g., angle of rotation), or simply G if the gate does not take parameters. For simplicity, we assume that gates have at most 1 parameter.

The semantics of a gate are defined in *path-sum* notation [Amy 2019] which shows how a gate transforms a basis state. From a verification perspective, a path sum is an expression defining the *transition relation*. Specifically, a single-qubit gate G^{ρ} is defined in the following fashion:

$$|x\rangle \to \sum_{y\in\mathbb{Z}_2} \phi(x,y,\rho)\,|f(x,y)\rangle \tag{1}$$

This path-sum specification says that for any basis state $|x\rangle$, applying G^{ρ} to $|x\rangle$ transforms the state into $\sum_{y\in\mathbb{Z}_2} \phi(x,y,\rho)\,|f(x,y)\rangle$, where

- $\phi \in \mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{C} \to \mathbb{C}$ is the *amplitude transformer*, and
- $f \in \mathbb{Z}_2 \times \mathbb{Z}_2 \to \mathbb{Z}_2$ is the *state transformer*.

Example 3.1. For Hadamard $H : |x\rangle \to \sum_{y\in\{0,1\}} \frac{1}{\sqrt{2}} e^{i\pi xy}\,|y\rangle$, $\phi(x,y) = \frac{1}{\sqrt{2}} e^{i\pi xy}$ and $f(x,y) = y$.

***n*-qubit gates.** n-qubit gates are analogously defined; a gate G^{ρ} has a path sum of the form

$$|\mathbf{x}\rangle \to \sum_{\mathbf{y}\in\mathbb{Z}_2^n} \phi(\mathbf{x},\mathbf{y},\rho)\,|f(\mathbf{x},\mathbf{y})\rangle$$

where $\phi \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n \times \mathbb{C} \to \mathbb{C}$ and $f \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n \to \mathbb{Z}_2^n$. We write $f(\mathbf{x},\mathbf{y})_i$ for the ith bit of $f(\mathbf{x},\mathbf{y})$.

$$\frac{[\![G^\rho]\!] \equiv \sum_{y\in\mathbb{Z}_2} \phi(x,y,\rho)\,|f(x,y)\rangle}{[\![G^\rho\ i]\!] \equiv \sum_{y\in\mathbb{Z}_2} \phi(x_i,y,\rho)\,|x_1\ldots x_{i-1} f(\mathbf{x},y) x_{i+1}\ldots x_n\rangle}\ \textsc{extend}$$

$$\frac{[\![C_1^{\boldsymbol{\rho}}]\!] \equiv \sum_{\mathbf{y_1}\in\mathbb{Z}_2^n} \phi_1(\mathbf{x},\mathbf{y_1},\boldsymbol{\rho})\,|f_1(\mathbf{x},\mathbf{y_1})\rangle \qquad [\![C_2^{\boldsymbol{\rho}}]\!] \equiv \sum_{\mathbf{y_2}\in\mathbb{Z}_2^n} \phi_2(\mathbf{x},\mathbf{y_2},\boldsymbol{\rho})\,|f_2(\mathbf{x},\mathbf{y_2})\rangle}{[\![C_1^\rho; C_2^\rho]\!] \equiv \sum_{\mathbf{y_1}\in\mathbb{Z}_2^n} \phi_1(\mathbf{x},\mathbf{y_1},\boldsymbol{\rho})([\![C_2^{\boldsymbol{\rho}}]\!][\mathbf{x}\leftarrow f_1(\mathbf{x},\mathbf{y_1})])}\ \textsc{seq}$$

Fig. 5. Path sum circuit semantics. $[\![C_2^{\boldsymbol{\rho}}]\!][\mathbf{x}\leftarrow f_1(\mathbf{x},\mathbf{y})]$ is $[\![C_2^{\boldsymbol{\rho}}]\!]$ with every instance of $\mathbf{x}$ replaced by $f_1(\mathbf{x},\mathbf{y})$.

Monomial gates. Some gates do not transform a basis state into *superposition* (non-trivial linear combination of basis states), and therefore their path-sums can be simplified as follows:

$$|\mathbf{x}\rangle \to \phi(\mathbf{x},\rho)\,|f(\mathbf{x})\rangle$$

We call such gates *monomial* gates because their matrix representation is a monomial matrix (or a generalized permutation matrix). E.g., CX is a monomial gate with path sum $|x_1x_2\rangle \to |x_1(x_1\oplus x_2)\rangle$.

Path sums are expressions. Observe how the right-hand side of $\to$ in a path sum is an expression of a quantum state parameterized by two variables, $\mathbf{x}$ and ρ. Henceforth, for a gate G^ρ, we shall use $[\![G^\rho]\!]$ to denote the expression on the right-hand side of its path sum.

Example 3.2. $[\![H]\!]$ is $\sum_{y\in\{0,1\}} \frac{1}{\sqrt{2}} e^{i\pi xy}\,|y\rangle$.

We will use the notation $[\![G_1^\rho]\!] \equiv [\![G_2^\rho]\!]$ to denote that $\forall \mathbf{x},\rho.\ [\![G_1^\rho]\!] = [\![G_2^\rho]\!]$. In other words, the two path sums define the same quantum state for every valuation of the variables $\mathbf{x}$ and ρ.

3.2 Circuit Semantics

Circuits. A quantum circuit over n qubits is a sequence of gates. Without loss of generality, we restrict ourselves to one- and two-qubit gates. We will write $G^\rho\ i$ to denote the single-qubit gate G^ρ applied to the ith qubit, and $G^\rho\ i\ j$ to denote the two-qubit gate G^ρ applied to the ith and jth qubits. A circuit C is defined by the following grammar:

$$C \;:=\; G^\rho\ i \;\mid\; G^\rho\ i\ j \;\mid\; C_1; C_2 \tag{2}$$

where $i,j\in[1,n]$. We will use $C^{\boldsymbol{\rho}}$ to denote that the circuit has a number of parameters, a vector $\boldsymbol{\rho}$ containing the parameters of the circuit's constituent gates.

Semantics of circuits. The path-sum representation of a circuit $[\![C^{\boldsymbol{\rho}}]\!]$ follows the grammar recursively: Either it is the path sum of a single gate, $[\![G^\rho\ i]\!]$ or $[\![G^\rho\ i\ j]\!]$, or the composition of the path sums of two circuits, $[\![C_1;C_2]\!]$. See Fig. 5 for the definitions.

Consider the rule EXTEND in Fig. 5. Given a single-qubit G^ρ gate, EXTEND defines the path sum to denote that G^ρ is applied to the ith qubit of an n-qubit system. Intuitively, the state transformer of $G^\rho\ i$ modifies the ith qubit, leaving the rest intact. EXTEND for two-qubit gates is analogous.

Example 3.3. Recall the Hadamard gate, where $[\![H]\!]$ is $\sum_{y\in\{0,1\}} \frac{1}{\sqrt{2}} e^{i\pi xy}\,|y\rangle$. Following the EXTEND rule, $[\![H\ i]\!]$ is $\sum_{y\in\{0,1\}} \frac{1}{\sqrt{2}} e^{i\pi x_i y}\,|x_1\ldots x_{i-1} y x_{i+1}\ldots x_n\rangle$.

The rule SEQ defines the semantics of composition. Informally, composing two path sums stitches together the "final basis states" of the first with the "initial basis states" of the second.

Example 3.4. Consider $[\![H]\!]$ and $[\![R_z^\theta]\!]$, which are $\sum_{y\in\{0,1\}} \frac{1}{\sqrt{2}} e^{i\pi xy} |y\rangle$ and $e^{i(2x-1)\theta} |x\rangle$, respectively. For the single-qubit circuit $H; R_z^\theta$, $[\![H; R_z^\theta]\!]$ is the following: $\sum_{y\in\{0,1\}} \frac{1}{\sqrt{2}} e^{i\pi xy} \underbrace{e^{i(2y-1)\theta} |y\rangle}_{[\![R_z^\theta]\!][x\leftarrow y]}$.

Circuit equivalence. Two cicuits are equivalent if they have equivalent path sums.

Definition 3.5 (Circuit equivalence). Consider two circuits $C_1^{\boldsymbol{\rho}}$ and $C_2^{\boldsymbol{\rho}}$ over the same set of parameters, $\boldsymbol{\rho}$. We say that the two circuits are equivalent iff $[\![C_1^{\boldsymbol{\rho}}]\!] \equiv [\![C_2^{\boldsymbol{\rho}}]\!]$.

Example 3.6. Consider the two equivalent single-qubit circuits, $R_z^{\theta_1}; R_z^{\theta_2}$ and $R_z^{\theta_1+\theta_2}$, We have

$$[\![R_z^{\theta_1}; R_z^{\theta_2}]\!] \equiv e^{i(2x-1)\theta_1} e^{i(2x-1)\theta_2} |x\rangle \qquad [\![R_z^{\theta_1+\theta_2}]\!] \equiv e^{i(2x-1)(\theta_1+\theta_2)} |x\rangle$$

It is easy to see that for any values of x and the parameters θ_1 and θ_2, we have $[\![R_z^{\theta_1}; R_z^{\theta_2}]\!] = [\![R_z^{\theta_1+\theta_2}]\!]$.

3.3 Symbolic Circuits

Symbolic gates. We will often use unknown gates in a circuit. We will treat those *symbolic gates* as path sums where the amplitude and state transformers are undefined (or *uninterpreted*). We will use S to refer to a symbolic gate, where $[\![S]\!]$ is of the form:

$$\phi^u(\mathbf{x}) \, |f^u(\mathbf{x})\rangle \tag{3}$$

where ϕ^u and f^u are uninterpreted. Observe that S is a monomial gate; this practical assumption helps us restrict the space of interpretations of the state transformers.

Symbolic circuits. When a circuit uses a symbolic gate, we will call it a *symbolic circuit*. Given two symbolic circuits, ideally, we would like to discover constraints on the uninterpreted amplitude and state transformers under which the two circuits are equivalent. We simplify this problem and only consider constraints on state transformers, f^u, treating the amplitude transformers, ϕ^u, as parameters of the circuit. The simplification is due to the fact that there are finitely many possible constraints on the Boolean function, f^u, while it is challenging to discover constraints on the complex-valued transformer, ϕ^u.

Interpretations of state transformers. We will use I to denote an interpretation of the uninterpreted state transformers in a symbolic circuit C. We will use $C(I)$ to denote C with all uninterpreted state transformers of symbolic gates replaced with their interpretation in I.

Example 3.7. Consider the symbolic gate S where $[\![S]\!] \equiv \phi^u(\mathbf{x}) \, |f^u(\mathbf{x})\rangle$. Let the interpretation I set $f^u(\mathbf{x})$ to the identity function. Then, $[\![S(I)]\!] \equiv \phi^u(\mathbf{x}) \, |\mathbf{x}\rangle$.

Definition 3.8 (Unifying interpretations). Consider two symbolic circuits $C_1^{\boldsymbol{\rho}}$ and $C_2^{\boldsymbol{\rho}}$ that use the same symbolic gates with amplitude transformers $\phi_1^u, \ldots, \phi_k^u$. We say that I is a *unifying interpretation* of the two circuits if

$$\forall \phi_1^u, \ldots, \phi_k^u, \mathbf{x}, \boldsymbol{\rho}. \, [\![C_1^{\boldsymbol{\rho}}(I)]\!] = [\![C_2^{\boldsymbol{\rho}}(I)]\!]. \tag{4}$$

Intuitively, a unifying interpretation I creates two circuits that are equivalent, following Definition 3.5. The idea is that we can treat the amplitude transformers as if they are circuit parameters.

Example 3.9. Recall the two circuits in Fig. 1c; call them C_1 and C_2. We have

$$[\![C_1]\!] \equiv e^{i(2x_1-1)\theta_1} \cdot \phi^u(\mathbf{x}) \cdot e^{i(2f^u(\mathbf{x})_2-1)\theta_2} |\mathbf{x}\rangle \qquad [\![C_2]\!] \equiv \phi^u(\mathbf{x}) \cdot e^{i(2f^u(\mathbf{x})_2-1)(\theta_1+\theta_2)} |\mathbf{x}\rangle$$

Consider the unifying interpretation I where $f^u(x_1x_2) = x_2x_1$.

$$[\![C_1(I)]\!] \equiv e^{i(2x_1-1)\theta_1} \cdot \phi^u(\mathbf{x}) \cdot e^{i(2x_1-1)\theta_2} |\mathbf{x}\rangle \qquad [\![C_2(I)]\!] \equiv \phi^u(\mathbf{x}) \cdot e^{i(2x_1-1)(\theta_1+\theta_2)} |\mathbf{x}\rangle$$

Observe that $[\![C_1(I)]\!] \equiv [\![C_2(I)]\!]$. However, e.g., $f^u(x_1x_2) = x_1x_2$ is *not* a unifying interpretation.

4 CIRCUIT EQUIVALENCE VERIFIER

We now present a fast approach for probabilistically verifying equivalence of two circuits, which will be key for synthesizing rewrite rules. We reduce the problem to a constrained form of *polynomial identity testing* (PIT), and demonstrate that it can be solved using a standard randomized algorithm for checking equivalence of polynomials. We begin with the foundations of polynomial identity testing.

4.1 Polynomial Identity Testing & Schwartz–Zippel

We are given two polynomials p_1 and p_2 over the same set of n complex-valued variables. We want to check if $p_1(\mathbf{v}) = p_2(\mathbf{v})$ for all values of $\mathbf{v} \in \mathbb{C}^n$, concisely denoted as $p_1 = p_2$. We will use d to denote the maximum degree of the two polynomials.

We can verify equivalence of p_1 and p_2 with high probability as follows.

(1) Let $R \subset \mathbb{C}$ be a finite subset of the complex numbers.
(2) Sample n independent values, $\alpha_1, \ldots, \alpha_n$, from the uniform distribution over R.
(3) Return True if $p_1(\boldsymbol{\alpha}) = p_2(\boldsymbol{\alpha})$; otherwise, return False.

The correctness of the above algorithm directly follows from the Schwartz–Zippel lemma [Motwani and Raghavan 1995, Ch. 7], which we adapt to our purposes here:

Theorem 4.1. *If $p_1 = p_2$, the algorithm returns True. If $p_1 \neq p_2$, the algorithm returns True (false positive) with probability at most $d/|R|$ (conversely, the algorithm returns False with probability at least $1 - d/|R|$).*

Proof. First case ($p_1 = p_2$): the algorithm returns True since for any $\boldsymbol{\alpha}$ we have $p_1(\boldsymbol{\alpha}) = p_2(\boldsymbol{\alpha})$. Second case ($p_1 \neq p_2$): The Schwartz–Zippel lemma says that if we sample $\alpha_1, \ldots, \alpha_n$ independently and uniformly from the finite set R, the probability that $p_1(\boldsymbol{\alpha}) = p_2(\boldsymbol{\alpha})$ is at most $d/|R|$. □

Observe that the algorithm has a small probability of a false positive: If the algorithm returns False, then we know that $p_1 \neq p_2$, since $\boldsymbol{\alpha}$ serves as a counterexample to equivalence. However, given $p_1 \neq p_2$, the algorithm may return the wrong answer (True) with a small probability. The probability of failure $d/|R|$ can be made arbitrarily small by sampling from a larger finite domain R.

Example 4.2. Suppose p_1 and p_2 are inequivalent, degree 10 polynomials. If we take R to be the set of 64-bit integers, we will have a failure probability on the order of 10^{-19}.

Constrained identity testing. A nice property of PIT is that we can readily apply it to checking equivalence of two polynomials under the constraint that *some* variables have a restricted domain in $\mathbb{C}$. This is critical in our setting, since we will have variables constrained to the unit circle, denoted $\mathbb{S} = \{c \in \mathbb{C} \mid |c| = 1\}$. Specifically, say we want to prove the following:

$$p_1(\mathbf{u}, \mathbf{v}) = p_2(\mathbf{u}, \mathbf{v}) \text{ for all } \mathbf{u} \in \mathbb{C}^n \text{ and } \mathbf{v} \in Z^m, \text{ where } Z \subset \mathbb{C} \tag{5}$$

Then, we can simply apply PIT by using a finite sample space $R \subseteq Z$.

Corollary 4.3 (Constrained PIT). *Consider Eq. (5). Apply PIT to check $p_1 = p_2$ with $R \subseteq Z$. If for all $\mathbf{u} \in \mathbb{C}^n$ and $\mathbf{v} \in Z^m$ we have $p_1(\mathbf{u}, \mathbf{v}) = p_2(\mathbf{u}, \mathbf{v})$, the algorithm returns True. Otherwise, if there exists $\mathbf{u} \in \mathbb{C}^n$ and $\mathbf{v} \in Z^m$ such that $p_1(\mathbf{u}, \mathbf{v}) \neq p_2(\mathbf{u}, \mathbf{v})$, the algorithm returns True with probability at most $d/|R|$.*

Proof. First case ($p_1 = p_2$): The algorithm will correctly return True because it samples each α_i from $R \subseteq Z \subset \mathbb{C}$, and we know from Eq. (5) that $p_1(\mathbf{u}, \mathbf{v}) = p_2(\mathbf{u}, \mathbf{v})$ for all $\mathbf{u} \in \mathbb{C}^n$ and $\mathbf{v} \in Z^m$. Second case ($p_1 \neq p_2$): following Schwartz–Zippel (Thm. 4.1), the algorithm returns True with probability $\leqslant d/|R|$. □

Observe how in the case the algorithm correctly returns True, you actually get a more general result than needed—a probabilistic guarantee that $p_1 = p_2$ with no constraints on the $\mathbf{v}$ variables.

4.2 Circuit-Equivalence Verification as Constrained Identity Testing

To prove equivalence of two circuits, we will check the equivalence of their amplitudes for every input basis state. While there are exponentially many amplitudes in the number of qubits, for synthesizing rewrite rules, we only care about circuits with a relatively small number of qubits, and so we do not suffer an exponential explosion. We will demonstrate that amplitude expressions are *constrained* polynomials, and therefore reduce the equivalence problem to PIT.

The following approach works for symbolic and non-symbolic pairs of circuits, under the assumption that all state transformers are interpreted. For simplicity, we assume that uninterpreted amplitude transformers are part of the circuit parameters.

Amplitude equivalence. Consider two circuits $C_1^{\boldsymbol{\rho}}$ and $C_2^{\boldsymbol{\rho}}$. Fix a constant $\mathbf{a} \in \mathbb{Z}_2^n$, which we will use as the initial basis state. We can write $[\![C_1^{\boldsymbol{\rho}}]\!][\mathbf{x} \leftarrow \mathbf{a}]$ and $[\![C_2^{\boldsymbol{\rho}}]\!][\mathbf{x} \leftarrow \mathbf{a}]$ as follows:

$$\sum_{\mathbf{y} \in \mathbb{Z}_2^n} \psi_1^{\mathbf{a}}(\mathbf{y}, \boldsymbol{\rho}) \, |\mathbf{y}\rangle \qquad\qquad \sum_{\mathbf{y} \in \mathbb{Z}_2^n} \psi_2^{\mathbf{a}}(\mathbf{y}, \boldsymbol{\rho}) \, |\mathbf{y}\rangle$$

Intuitively, $\psi_1^{\mathbf{a}}(\mathbf{y}, \boldsymbol{\rho})$ is an expression of the amplitude of basis state $\mathbf{y}$ if we apply $C_1^{\boldsymbol{\rho}}$ to state $|\mathbf{a}\rangle$.

Example 4.4. From Example 3.4, $[\![H; R_z^\theta]\!]$ is $\sum_{y \in \{0,1\}} \underbrace{\frac{1}{\sqrt{2}} e^{i\pi xy} e^{i(2y-1)\theta}}_{\psi^x(y,\theta)} |y\rangle$.

The following lemma reframes circuit equivalence as checking the equality of amplitudes:

LEMMA 4.5. *$C_1^{\boldsymbol{\rho}}$ and $C_2^{\boldsymbol{\rho}}$ are equivalent iff for all $\mathbf{x}$, $\mathbf{y}$, and $\boldsymbol{\rho}$, $\psi_1^{\mathbf{x}}(\mathbf{y}, \boldsymbol{\rho}) = \psi_2^{\mathbf{x}}(\mathbf{y}, \boldsymbol{\rho})$.*

We can eliminate the universal quantifier over $\mathbf{x}$ and $\mathbf{y}$ in Lem. 4.5 by turning it into a finite summation, following the basic arithmetic fact:

$$\text{If } \alpha = \beta \text{ and } \alpha' = \beta', \text{ then } z\alpha + z'\alpha' = z\beta + z'\beta' \text{ for all } z, z'.$$

THEOREM 4.6. *For every $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_2^n$, create a fresh complex-valued variable $v_{\mathbf{a},\mathbf{b}}$. Then, $C_1^{\boldsymbol{\rho}}$ and $C_2^{\boldsymbol{\rho}}$ are equivalent iff for all values of the parameters $\boldsymbol{\rho}$ and the fresh variables,*

$$\sum_{\mathbf{a},\mathbf{b} \in \mathbb{Z}_2^n} v_{\mathbf{a},\mathbf{b}} \cdot \psi_1^{\mathbf{a}}(\mathbf{b}, \boldsymbol{\rho}) = \sum_{\mathbf{a},\mathbf{b} \in \mathbb{Z}_2^n} v_{\mathbf{a},\mathbf{b}} \cdot \psi_2^{\mathbf{a}}(\mathbf{b}, \boldsymbol{\rho}) \tag{6}$$

Observe that Eq. (6) is only over the freshly introduced (v) variables and the parameters $\boldsymbol{\rho}$.

Example 4.7. Continuing Example 4.4, if $H; R_z^\theta$ is circuit C_1 in Thm. 4.6, then the left-hand side of Eq. (6) will be $\sum_{a,b} v_{a,b} \underbrace{\frac{1}{\sqrt{2}} e^{i\pi ab} e^{i(2b-1)\theta}}_{\psi^{\mathbf{a}}(b,\theta)} |b\rangle$. After expanding:

$$\left(v_{0,0} \cdot \frac{1}{\sqrt{2}} \cdot e^{-i\theta}\right) + \left(v_{0,1} \cdot \frac{1}{\sqrt{2}} \cdot e^{i\theta}\right) + \left(v_{1,0} \cdot \frac{1}{\sqrt{2}} \cdot e^{-i\theta}\right) + \left(v_{1,1} \cdot \frac{e^{i\pi}}{\sqrt{2}} \cdot e^{i\theta}\right)$$

Amplitudes are polynomials. We make the observation that for all quantum gates of interest, the two sides of Eq. (6) can be reduced to constrained polynomials. Therefore, we can reduce checking Eq. (6) to constrained PIT. Specifically, each side of Eq. (6) can be written in the form

$$\sum_j c_j \prod_k t_k$$

where c_j is a constant and t_k is a term that can have three different forms:

(1) variables $v_{\mathbf{a},\mathbf{b}}$, introduced in Thm. 4.6,
(2) $(\phi^u(\mathbf{a},\mathbf{b}))^n$, where $n \in \mathbb{N}$,
(3) or $(e^{i\theta})^n$, where θ is a parameter of the circuit and $n \in \mathbb{Z}$.

Observe therefore that Eq. (6) very much resembles a polynomial over complex variables, but only one of the three kinds of terms t_k is a complex variable ($v_{\mathbf{a},\mathbf{b}}$). We now show how to transform the rest of the terms into complex variables.

- First, applications of amplitude transformers, $\phi^u(\mathbf{a},\mathbf{b})$, of symbolic gates. Since the domain of ϕ^u is finite, we replace each application of the form $\phi^u(\mathbf{a},\mathbf{b})$ with a complex variable $\phi^u_{\mathbf{a},\mathbf{b}}$.
- Second, terms of the form $e^{i\theta}$, which come from the gates in the circuit. Note that $e^{i\theta}$ is a point on the complex unit circle, $\mathbb{S}$, parameterized by the angle θ. Therefore, for every unique term $e^{i\theta}$, we replace $e^{i\theta}$ with a complex-valued variable v_θ and constrain it to $\mathbb{S}$.[1]

The above transformation reduces the problem in Thm. 4.6 to constrained PIT, which can be solved using Schwartz–Zippel (Cor. 4.3).

Theorem 4.8 (Reduction to constrained PIT). *In the context of Thm. 4.6, assume that Eq. (6) has k unique terms of the form $e^{i\theta_1}, \ldots e^{i\theta_k}$. Apply the above transformation to Eq. (6) and let $p_1 = p_2$ be the resulting equality. Then, C_1^ρ is equivalent to C_2^ρ iff $p_1 = p_2$ under the constraint that $v_{\theta_1}, \ldots, v_{\theta_k} \in \mathbb{S}$.*

Example 4.9. Suppose we want to check the following equality: $e^{i\theta_1} \cdot \phi^u(00) \cdot e^{i\theta_2} = 0$. We transform $\phi^u(00)$ into a fresh variable ϕ^u_{00} and $e^{i\theta_1}$ and $e^{i\theta_2}$ into v_{θ_1} and v_{θ_2}, respectively. This results in the following constrained PIT problem: $v_{\theta_1} \cdot \phi^u_{00} \cdot v_{\theta_2} = 0$, for all $v_{\theta_1}, v_{\theta_2} \in \mathbb{S}$ and $\phi^u_{00} \in \mathbb{C}$.

5 REWRITE-RULE SYNTHESIZER

We now present our rewrite-rule synthesizer. The naïve approach is to enumerate pairs of circuits and check their equivalence—a quadratic explosion. To avoid this quadratic explosion, we will utilize a new probabilistic data structure in which circuits are inserted and stored in their respective equivalence classes. We call this data structure a *polynomial identity filter* (PIF), because it uses the high-probability guarantees of Schwartz–Zippel to populate circuits into equivalence classes.

5.1 The Polynomial Identity Filter (PIF)

We will now define the polynomial identity filter (PIF). Our goal is to design a data structure that groups polynomials into equivalence classes; when a new polynomial is *inserted*, it will assign it to the appropriate equivalence class. The PIF directly builds upon the insights of Schwartz–Zippel.

The key trick of the PIF is to randomly sample $\boldsymbol{\alpha}$ *only once* at initialization and use it to compare all inserted polynomials. Building upon the high-probability guarantees of Schwartz–Zippel, we ensure that all deduced equivalences are correct with a high probability.

Initialization. To define equivalence classes of polynomials, we will use a map M from complex numbers to *sets of polynomials*. We initialize our polynomial identity filter as follows:

(1) Let M map every complex number to the empty set.
(2) Let $R \subset \mathbb{C}$ be a finite subset of the complex numbers.
(3) Sample n independent values, $\alpha_1, \ldots, \alpha_n$, from the uniform distribution over R. *These values are sampled once initially and used throughout the lifetime of the data structure.*

[1] We can handle terms with negative exponents like $e^{-i\theta}$ by multiplying both polynomials by $e^{i\theta}$. Terms with expressions such as $e^{i(\theta_1+\theta_2)}$ can be expanded to $e^{i\theta_1}e^{i\theta_2}$.

Inserting a polynomial into PIF. When a new polynomial p is inserted into the PIF, we update the map by adding p to the set $M[p(\boldsymbol{\alpha})]$. Intuitively, the set $M[p(\boldsymbol{\alpha})]$ is the set of all polynomials that evaluate to the same complex number on the input $\boldsymbol{\alpha}$.

Correctness guarantees. Suppose we insert ℓ polynomials into the PIF. Intuitively, the PIF implicitly applies the PIT algorithm from § 4.1 to all pairs of polynomials, i.e., $\leqslant \ell^2$ polynomial identity checks. Since every identity test has a false positive probability (if the polynomials are not equal), the total failure probability of the PIF increases. Luckily, the high-probability guarantees of Schwartz–Zippel still provide us with a pretty good failure probability. Simply following the union bound, the failure probabilities add up, as formalized in the following theorem:

THEOREM 5.1 (PIF WORST-CASE GUARANTEES). *Suppose we insert ℓ mutually inequivalent polynomials into a new PIF. Let d be the maximum degree of all ℓ polynomials. The probability that one of the cells of M contains more than one polynomial is at most $\ell^2 d/|R|$.*

PROOF. Let $p_1, \ldots, p_\ell$ be mutually inequivalent polynomials over the same set of n variables. If we insert ℓ polynomials into the PIF, it implicitly performs the following randomized computation:

(1) Sample $\alpha_1, \ldots, \alpha_n$ independently and uniformly from R.
(2) For every pair p_i and p_j, where $i \neq j$, check if $p_i(\boldsymbol{\alpha}) = p_j(\boldsymbol{\alpha})$.

(Note that step 2 is performed efficiently by computing each $p_i(\boldsymbol{\alpha})$ separately and inserting p_i into $M[p_i(\boldsymbol{\alpha})]$. All p_i, p_j such that $p_i(\boldsymbol{\alpha}) = p_j(\boldsymbol{\alpha})$ are therefore inserted into the same cell of M.)

Following Schwartz–Zippel (Thm. 4.1), for any pair of polynomials p_i and p_j, where $i \neq j$, $\Pr[p_i(\boldsymbol{\alpha}) = p_j(\boldsymbol{\alpha})] \leqslant d/|R|$. Therefore, following the union bound, the probability that one of the cells of M contains more than one polynomial is

$$\Pr[\exists i \neq j.\, p_i(\boldsymbol{\alpha}) = p_j(\boldsymbol{\alpha})] \leqslant \sum_{i,j \in [1,\ell], i \neq j} \Pr[p_i(\boldsymbol{\alpha}) = p_j(\boldsymbol{\alpha})] \leqslant \ell^2 d/|R|$$

□

Example 5.2. If we insert 10^6 mutually inequivalent polynomials of degree 10 into a new PIF, and we use 64-bit integers for R, then the probability of the PIF declaring a pair equivalent is $\sim 10^{-7}$.

Implementation considerations. To further minimize failure probability, if necessary, after populating the data structure, we can apply PIT (with freshly sampled values) to each pair of polynomials in each equivalence class. An equivalence class will typically contain a small number of polynomials, allowing us to enumerate all pairs and reverify their equivalence. To avoid floating-point errors, we can restrict our sample space R to rational numbers (see full paper for details).

5.2 Rewrite-Rule Synthesizer

We now have all the ingredients needed to describe our rewrite-rule synthesis technique.

Symbolic-circuit grammar. Fig. 6 shows the grammar of symbolic circuits that we consider. We fix finite sets of one- and two-qubit gates, symbolic gates, parameter variables (θ), and constants. We also allow for arithmetic expressions over parameters, e.g., $\theta_1 + \theta_2$.

The synthesis algorithm. Alg. 1 synthesizes pairs of equivalent circuits. It starts with a PIF instance containing the empty circuit. Then, in a bottom-up-synthesis fashion, the algorithm enumerates circuits of increasing size, up to a bound k, and inserts them into the PIF. We assume that all circuits are transformed into polynomials, following Thm. 4.8, before inserting them into the PIF. We also assume that the polynomials all share the same $v_{\mathbf{x},\mathbf{y}}$ variables from Thm. 4.8.

For symbolic circuits, the algorithm considers every possible interpretation of the symbolic gates' state transformers (f^u). Note that the space of interpretations is restricted to reversible functions because quantum operations are reversible.

Algorithm 1 Circuit equivalence synthesizer

```
procedure SYNTH-EQ
    Construct a PIF instance and insert the empty circuit
    Let 𝓕 be the space of all reversible functions in ℤ_2^n → ℤ_2^n
    Let C be all circuits with size up to some fixed bound, following grammar in Fig. 6
    for C ∈ C do                                          ▷ in order of increasing size
        if C contains no symbolic gates then insert C into the PIF
        else
            Let f_1, ..., f_l be the uninterpreted state transfomers in C
            for every interpretation I of f_1, ..., f_l from 𝓕 do
                insert C(I) into the PIF
```

After Alg. 1 completes, we take each equivalence class in the PIF and generate a set of rules. For every equivalent pair of circuits, (C_1, C_2), where C_2 is smaller than C_1 (by number of gates), we generate the rewrite rule $C_1 \longrightarrow C_2$. We call these *size-reducing* rules. For equivalent pairs of the same size, we generate $C_1 \longrightarrow C_2$ and $C_2 \longrightarrow C_1$. We call these *size-preserving* rules. For symbolic circuits, we construct rewrite rules where the two circuits have the same interpretation.

$$
\begin{array}{rll}
C & \coloneqq G^{\rho}_{1,1}\ i \mid G^{\rho}_{1,2}\ i \mid \ldots & \text{1-qubit gates} \\
& \mid G^{\rho}_{2,1}\ i\ j \mid G^{\rho}_{2,2}\ i\ j \mid \ldots & \text{2-qubit gates} \\
& \mid S_1 \mid S_2 \mid \ldots & \text{symbolic gates} \\
& \mid C_1; C_2 & \text{sequential comp.} \\
\rho & \coloneqq \theta_1 \mid \theta_2 \mid \ldots \mid -\rho \mid c\rho \mid \rho + \rho & \text{parameter expr.} \\
i, j & \in [1, n] & \text{qubit indices} \\
c & \in \{\pi, -\pi, \pi/2, \ldots\} & \text{constants}
\end{array}
$$

Fig. 6. Circuit synthesis grammar

Pruning techniques. To prune unnecessary rules, we adopt two techniques from Quartz [Xu et al. 2022a]: (1) Picking a representative circuit from each equivalence class to construct larger circuits with the grammar (any equivalent circuit can be rewritten to the representative and vice versa). (2) Prune rules where both sides have common subcircuits. We also incorporate some new heuristics such as pruning rules where the left-hand side contains functions in parameter expressions, e.g., $\theta_1 + \theta_2$. The complete list of additional pruning we perform is described in the full paper.

6 CIRCUIT OPTIMIZER

Given a circuit, we apply the synthesized rewrite rules to minimize some cost function: Commonly, this is the number of gates in the circuit because each gate, particularly two-qubit gates, introduces noise in the computation. There are two critical challenges here:

(1) How do we apply symbolic rules that can match arbitrary subcircuits? While there are standard algorithms for finding patterns in a quantum circuit, there are no general techniques for finding patterns that satisfy a given constraint.
(2) In what order to apply the rules? Optimizers, like VOQC and TKET, employ a fixed schedule of optimizations chosen by the compiler designer. QUESO synthesizes tens of thousands of rules, and we simply cannot ask a developer to experiment with different schedules.[2]

To address these challenges, we present (1) an algorithm for matching and applying symbolic rewrite rules, and (2) a beam-search-based optimization algorithm.

6.1 Rule-Matching Algorithm

Given a quantum circuit C and a rewrite rule $C_l \longrightarrow C_r$, we want to find subcircuits of C that match the pattern C_l, and rewrite them to C_r. For non-symbolic rewrite rules, this is a standard process.

[2]Equality saturation, as realized in the state-of-the-art library, egg [Willsey et al. 2021], cannot scale to large numbers of rules, especially *multi-pattern* ones, and cannot apply symbolic rules natively.

Algorithm 2 Maximal beam search

```
procedure MAX-BEAM(C)
    Create priority queue Q of bounded size, and add C to Q
    C_best ← C
    while Q is not empty do
        dequeue circuit C'
        if COST(C') < COST(C_best) then
            C_best ← C'
        for every rewrite rule R do
            C'_R ← APPLY-MAX(R, C')
            if COST(C'_R) ⩽ COST(C_best) and C'_R has not been seen before then
                add C'_R to Q
    return C_best
```

First, a quantum circuit is represented as a directed-acyclic graph (DAG), just like in our graphical representations of circuits in, e.g., Fig. 1. Finding the pattern C_l in C boils down to the following problem: Find a subgraph in C that is *isomorphic* to C_l.[3] Since this is a well-known problem, we use $\text{MATCH}(C_l, C)$ to denote the procedure that returns *all* subgraphs in C that match the pattern C_l.

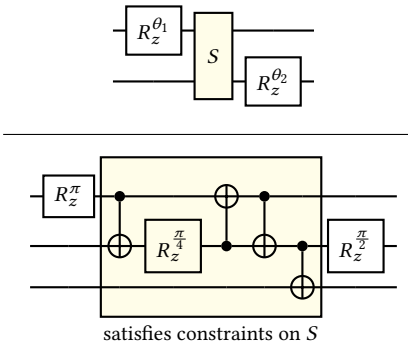


Fig. 7. Example of MATCH-SYMB

Matching symbolic patterns. For simplicity, and without loss of generality, we consider symbolic rewrite rules that contain a single symbolic gate S. We fix a rule of the form $C_l; S; C_l' \longrightarrow C_r; S; C_r'$, where the state transformer of S is interpreted by I. We want to formalize matching the pattern $C_l; S; C_l'$ in a circuit C. We assume that $[\![S(I)]\!]$ is of the form $\phi^u(\mathbf{x})\,|f(\mathbf{x})\rangle$ and that the circuit C has n qubits. The idea is that we will have to try every possible circuit that matches the path sum of S, as formalized in MATCH-SYM:

$$\text{MATCH-SYM}(C_l; S, C_l',\ C) = \bigcup_{C_S \in \mathcal{S}} \text{MATCH}(C_l; C_S; C_l',\ C) \tag{7}$$

where $\mathcal{S} = \{C_S \mid C_S \text{ is a non-symbolic } n\text{-qubit circuit and } [\![C_S]\!] \text{ is of the form } \phi(\mathbf{x}\ldots)\,|f(\mathbf{x})\ldots\rangle\}$. Observe that the circuits C_S can apply operations to more qubits than in S, (as indicated by the $\ldots$). As formalized in Thm. 6.2, this procedure preserves the correctness of the rewrite rule.

Example 6.1. Consider the 2-qubit symbolic pattern in Fig. 7 (top), where $[\![S(I)]\!] \equiv \phi^u(x_1x_2)\,|x_2x_1\rangle$. The 3-qubit circuit in Fig. 7 (bottom) matches the symbolic pattern. The highlighted subcircuit has a path sum of the form $e^{i(2x_2-1)\pi/4}\,|x_2x_1\ldots\rangle$, which matches $[\![S(I)]\!]$, because it swaps the first two qubits, x_1 and x_2.

THEOREM 6.2 (SOUNDNESS OF MATCH-SYM). *Given a symbolic rewrite rule of the form $C_l; S; C_l' \longrightarrow C_r; S; C_r'$ and $C_S \in \mathcal{S}$, $C_l; C_S; C_l' \equiv C_r; C_S; C_r'$.*

Implementing MATCH-SYM. We implement MATCH-SYM by restricting $\mathcal{S}$ to the space of subcircuits of C. For efficiency, we limit $\mathcal{S}$ by (1) only considering subcircuits of C over monomial gates, since S is monomial, and (2) limiting the search to subcircuits *between* the set of subcircuits that match C_l and C_l'. Additionally (see § 7) we limit the size of circuits in $\mathcal{S}$. Our approach for checking if a subcircuit is monomial is inspired by Nam et al. [2018]'s rotation-merging implementation.

[3] additionally the subgraph has to be *convex*.

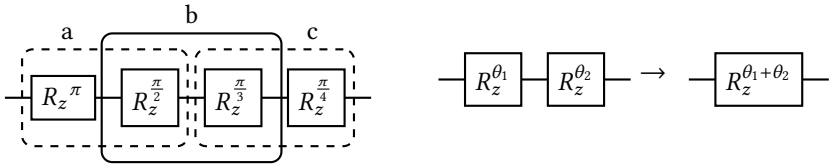


Fig. 8. Maximal match example

6.2 Maximal Beam Search

To find an optimal circuit, one needs to exhaustively consider every possible ordering of rewrite rule application. To limit the combinatorial explosion, the scheduling algorithm we propose, MAX-BEAM (Alg. 2), limits the size of the search space in two ways: (1) Instead of considering a single application of a rewrite rule in each step of the search, MAX-BEAM greedily considers *maximal* applications of a rewrite rule. (2) MAX-BEAM is a *beam search* through the space of rewrites.

Definition 6.3 (Maximal matching set). Consider a rewrite rule $C_l \longrightarrow C_r$ and a circuit C. A *maximal matching* set is a subset $\mathcal{M} \subseteq \text{MATCH}(C_l, C)$ such that (1) no pair of subcircuits $C_l', C_l'' \in \mathcal{M}$ overlap in C, and (2) there is no $\mathcal{M}'$, where $\mathcal{M} \subset \mathcal{M}' \subseteq \text{MATCH}(C_l, C)$, that satisfies condition (1). The same definition applies to symbolic rules.

Example 6.4. Consider the single-qubit circuit with a sequence of four rotations in Fig. 8 (left) and the rewrite rule that merges two rotations (right). The rule matches three subcircuits as shown by the three boxes. Matches (a) and (b) overlap, as well as (b) and (c). So APPLY-MAX chooses a set of matches that do not overlap—e.g., the dotted ones—and applies the rewrite to them.

The MAX-BEAM algorithm begins with a priority queue of fixed size containing the input circuit C that we wish to optimize. The priority queue uses a cost function, COST. The algorithm picks the next circuit from Q and rewrites it. For every rewrite rule R, it applies R *maximally* to the current circuit C'. Specifically, the function APPLY-MAX finds a maximal set of *non-overlapping* matches for the rewrite rule R in C' and rewrites all the matches, producing a new circuit C_R'. In practice, we implement APPLY-MAX greedily and do not try to find a *maximum* matching set, only a maximal one. MAX-BEAM can be terminated after a finite number of iterations or within some time limit.

7 IMPLEMENTATION AND EVALUATION

Synthesized optimizers. We implemented QUESO in ~3,700 lines of Java. We evaluated QUESO on four different gate sets: (1) the standard gate set for IBM computers, (2) the gate set for Rigetti computers, (3) the gate set for *ion trap* computers (like IonQ [IonQ 2022b]), and (4) the gate set of Nam et al. [2018] (henceforth, Nam). The IBM and Rigetti gate sets support devices with superconducting qubits, which are the largest quantum devices physically realized so far. Ion trap architectures are attractive due to their all-to-all qubit connectivity, which reduces the need for expensive *swaps*. The Nam gate set is interesting to study because it closely resembles the *Clifford+T* universal gate set where Clifford gates can be efficiently simulated on a classical computer. However, unlike the other gate sets, the Nam gate set is not physically realized in any quantum hardware.

Table 1 summarizes the gate sets and the rules synthesized. For all gate sets, we limit rewrite rules to be over a maximum of 3 qubits and vary the maximum *size* of a rule: the number of gates on either side. We choose the largest size for which QUESO can synthesize rules within 3 minutes. For example, for IBM, in 72 seconds QUESO synthesizes 701 rules, out of which 48 rules are symbolic. Failure probability is an upper bound on the PIF returning an incorrect rule (Thm. 5.1). Observe how vanishingly small the failure probabilities are, e.g., 10^{-17} for IBM.

Table 1. Rewrite-rule synthesis results

Gate set	Gates	# Qubits	Size	# Possible Rules	# Rules	# Symbolic Rules	Failure Prob.	Time (s)
IBM	$U1^{\theta}$, $U2^{\theta_1,\theta_2}$, $U3^{\theta_1,\theta_2,\theta_3}$, CX	3	4	1.5×10^{13}	701	48	10^{-17}	72
Nam	H, X, R_z^{θ}, CX	3	6	5.4×10^{18}	14,544	2,365	10^{-12}	135
Rigetti	R_x^{π}, $R_x^{\pi/2}$, $R_x^{-\pi/2}$, R_z^{θ}, CZ	3	5	1.4×10^{16}	2,242	809	10^{-14}	70
Ion	R_x^{θ}, R_y^{θ}, R_z^{θ}, R_{xx}^{θ}	3	3	1.6×10^{11}	1,519	24	10^{-19}	15

Research questions. We aim to answer the following research questions:

(Q1) How does QUESO compare to state-of-the-art optimizers?

(Q2) How does QUESO compare to superoptimization?

(Q3) Which synthesized rewrite rules are useful?

Benchmarks. Throughout, we will use a set of 33 benchmark circuits, comprised of those from prior work on optimization [Amy et al. 2014; Hietala et al. 2021; Nam et al. 2018; Xu et al. 2022a] and a new class of circuits. The benchmarks from prior work include arithmetic circuits and Toffoli gate networks. We added *quantum approximate optimization algorithm* (QAOA) circuits that approximate the maximum cut in a 3-regular graph. QAOA is a promising and near-term application because it can approximate NP-hard combinatorial problems on NISQ machines without error correction.

Instantiation of QUESO. We use the total number of gates in a circuit as QUESO's cost function (COST in Alg. 2). We fix the priority queue size (in Alg. 2) to 8000 circuits. For matching symbolic circuits, we limit the number of qubits and size of the C_S circuits in Eq. (7) to 7 and 10, respectively.

Metrics. To compare tools, the main metric we use is the number of two-qubit gates, because they have *orders of magnitude* higher error rates compared to single-qubit gates. For instance, the error rates for single- and two-qubit gates on the IBM Toronto device are on the order of 10^{-4} and 10^{-2}, respectively [IBM 2022] (further, some single-qubit gates, like R_z, are error-free as they are simulated classically). To further illustrate the importance of two-qubit gate reduction, we use *fidelity* results (success probability), which we statically estimate based on publicly available error rates from the IBM Toronto [IBM 2022], Rigetti Aspen-11 [Rigetti 2022], and IonQ Aria [IonQ 2022a] devices. Fidelity is the probability that none of the gates in a circuit cause an error. For a circuit $G_1; \ldots; G_n$, its fidelity is $\prod_i (1 -$ error rate of $G_i)$.

Q1: How does QUESO compare to state-of-the-art optimizers?

Experimental setup. We compared QUESO to four state-of-the-art optimizers: IBM Qiskit [Aleksandrowicz et al. 2019], Quilc [Smith et al. 2020], TKET [Sivarajah et al. 2020], and VOQC [Hietala et al. 2021]. The first three are used in industrial toolkits; VOQC is a formally verified and very effective optimizer.[4] For each benchmark, we set a time limit of 1 hour (we discuss running time of QUESO in Q3). To ensure a fair comparison of the optimization phases of the various tools, we provide all tools with the same decomposed input circuit in the target gate set.

We use *S-curves* to present the results. For each benchmark circuit, we compute the quantity

$$\frac{\#\text{ of gates with tool } X - \#\text{ of gates with QUESO}}{\#\text{ of gates in unoptimized circuit}}$$

and present the benchmarks in increasing order. Positive values imply that QUESO outperforms tool X. We compute the following quantity for fidelity:[5] $\frac{\text{fidelity with QUESO} - \text{fidelity with tool } X}{\text{maximum of fidelity with QUESO and with tool } X}$.

[4] We excluded the Nam et al. [2018] optimizer because it is proprietary and the existing data was not obtained from running on decomposed input circuits nor does it include the added benchmarks. We also excluded PyZX [Kissinger and van de Wetering 2019a] because it works well for reducing T gate count, which is useful for future fault-tolerant machines, but can often increase total gate count. A comparison of QUESO against PyZX with respect to T gate reduction is in the full paper.

[5] We do not use the fidelity of the original circuit in the denominator here because it can be extremely small.

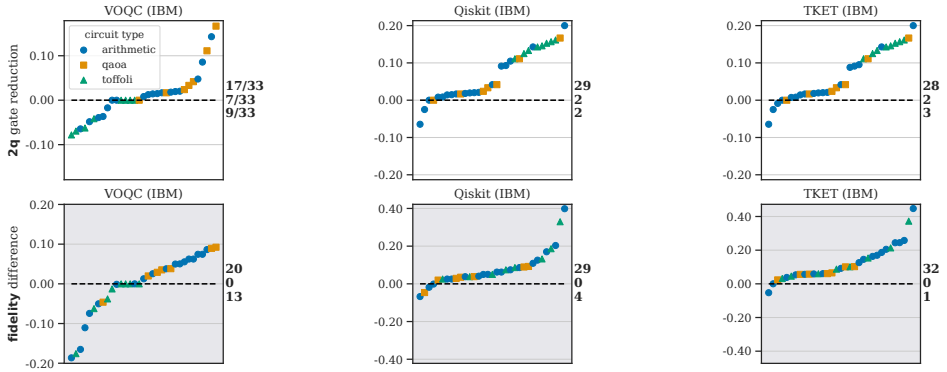


Fig. 9. Comparison against state-of-the-art optimizers on IBM. Each graph is annotated to the right with the number of circuits where QUESO outperforms, matches, and underperforms (top-to-bottom) the other tool.

IBM. Fig. 9 shows the S-curves for the IBM gate set. Consider, for instance, the top middle S-curve, which compares QUESO to IBM Qiskit. For the majority of the benchmarks, 29/33, QUESO outperforms Qiskit in two-qubit gate reduction—all the benchmarks above the dashed horizontal (0) line. We see similar results with TKET. QUESO can outperform VOQC in 17/33 benchmarks, and exactly match its performance on 7/33 benchmarks. The fidelity graphs, bottom row, depict a very similar story, indicating the close correspondence between two-qubit-gate count and fidelity.

Nam. Results on Nam are in the full paper because they resemble the results for the IBM gate set.

Rigetti and Ion. For the Rigetti and Ion gate sets, we compare against Quilc and Qiskit, respectively. Implemented by Rigetti, Quilc is specialized for optimizing the Rigetti gate set. To our knowledge, these are the only publicly available compilers that apply to those two gate sets. We also compare against TKET for the Rigetti gate set but we note that the optimized circuits TKET produces *do not* adhere to the allowed angles for R_x gates and therefore are not valid. See full paper for the results, which resemble the results for Quilc.

None of the tools are able to reduce two-qubit gate count for the majority of the benchmarks as shown in Fig. 10, where these benchmarks lay on the dashed line. However, we see reduction in single-qubit gates (see full paper), which is reflected in fidelity, as single-qubit gates errors dominate for all but 7/33 benchmarks. QUESO is able to outperform Quilc on a majority of the benchmarks for the Rigetti gate set (20/33). For the Ion gate set, we see an opposite story: QUESO outperforms Qiskit on 13/33 benchmarks and underperforms it on 20/33.

We investigated why Quilc and Qiskit are sometimes able to achieve reduction in two-qubit gates and isolated it to a powerful optimization that resynthesizes arbitrary two-qubit circuits [Cross et al. 2019]. We cannot fully capture such optimizations and leave it as an avenue for future work.

Q1 summary. **QUESO is able to significantly outperform or match state-of-the-art optimizers on a majority of the benchmarks across all gate sets with respect to two-qubit gate reduction. The results are similar for fidelity except on the Ion gate set where QUESO only outperforms or matches Qiskit on 39% of the benchmarks.**

Q2: How does QUESO compare to superoptimization?

Experimental setup. Next, we compare against the Quartz *superoptimizer* [Xu et al. 2022a]. Quartz is comprised of two phases that run in sequence: (1) The *preprocessing* phase: a manually written set of optimizations that decompose a circuit to the target gate set and applies rotation merging and other domain specific optimizations—e.g., Hadamard and CZ cancellation for Rigetti.

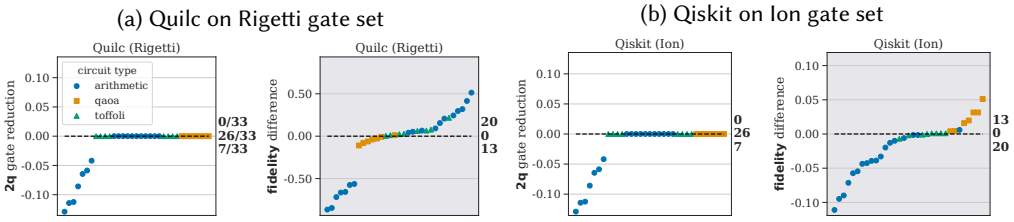


Fig. 10. Comparison against state-of-the-art optimizers on Rigetti and Ion trap.

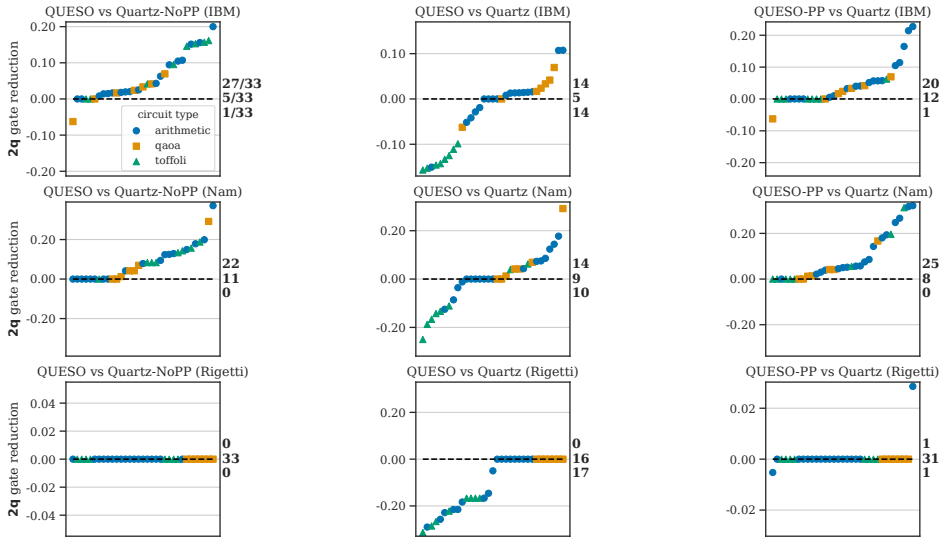


Fig. 11. **1 hour timeout** comparison against Quartz: (left column) Quartz without the preprocessing phase; (middle column) Quartz with both phases; (right column) QUESO with Quartz's preprocessing phase vs Quartz with both phases.

(2) The *search* phase: applies automatically synthesized (non-symbolic) rewrite rules and applies them to a circuit by enumerating different orderings. The synthesized rules are verified with an SMT solver, but the manually written preprocessing phase is not verified.

We compare against Quartz along two dimensions: (1) the time to synthesize rules and (2) the quality of the synthesized rules. Both tools were alotted 1 hour of optimization time, 32GB of RAM, and 1 CPU core per benchmark. We do not compare against Quartz for the Ion gate set, which they do not currently support. To fairly compare against Quartz, we distinguish between two variants of the tool: (1) Quartz, the full tool with the two phases, and (2) Quartz-NoPP, which is Quartz without the preprocessing phase—just the synthesized rules.

Results. First, we observe that QUESO is able to synthesize rules an order of magnitude faster than Quartz: QUESO synthesizes rules in 70-135 seconds, whereas Quartz takes up to 2,303 seconds for rules of the same size. For example, QUESO synthesizes rules for the IBM gate set with up to 3 qubits and size 4 in 72 seconds while Quartz takes 2,193 seconds. Note that comparing rule-synthesis speed directly is challenging because QUESO synthesizes more expressive, symbolic rules. We attribute our fast rule synthesis to using a PIF for equivalence checking rather than an SMT solver.

Fig. 11 shows the results of the comparison to Quartz. The first column shows that QUESO is able to significantly outperform Quartz with preprocessing disabled (Quartz-NoPP) on most benchmarks

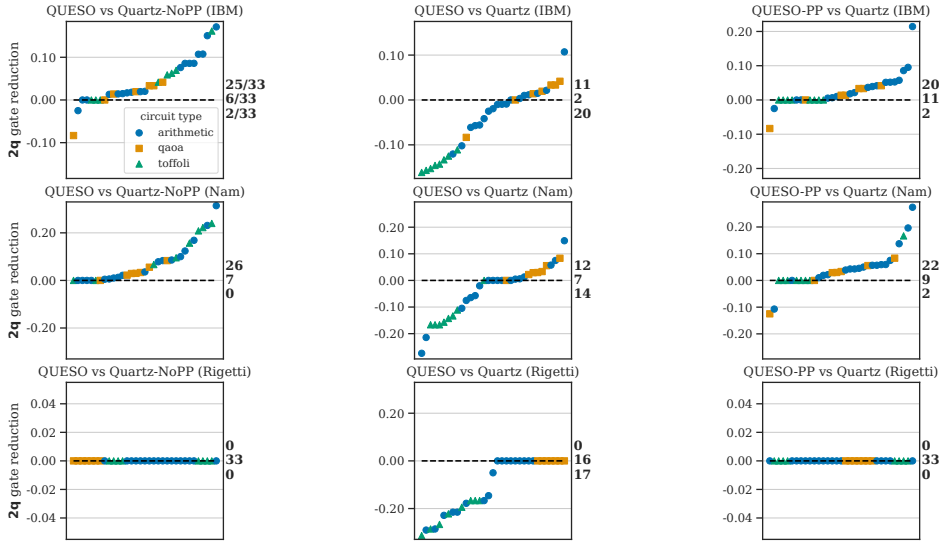


Fig. 12. **24 hour timeout** comparison against Quartz

on IBM and Nam gate sets. For Rigetti, both QUESO and Quartz-NoPP cannot eliminate any two-qubit gates. These results demonstrates the power of our synthesized rules compared to Quartz's.

Fig. 11 (middle column) shows the results against full Quartz, i.e., when enabling the hand-crafted preprocessing phase. Note that in this case we provide Quartz with circuits pre-decomposition—i.e., with Toffoli gates. For QUESO, we use the same decomposition for each Toffoli gate, whereas Quartz's preprocessing greedily picks which decomposition to use. The results on IBM and Nam show QUESO and Quartz are close in performance; for Rigetti, thanks to the domain-specific optimizations in the preprocessing phase, Quartz is able to eliminate two-qubit gates in half of the benchmarks.

To further understand the effects of the preprocessing phase, QUESO-PP is the result of running QUESO on the output of Quartz's preprocessing phase. As the third column of Fig. 11 shows, on IBM and Nam, QUESO-PP outperforms full Quartz, and matches it on Rigetti. These results further amplify the power of our symbolic rules in comparison with Quartz's.

The results are similar when running both tools for the full 24 hour timeout used in Quartz's original evaluation as shown in Fig. 12. The results for every one hour interval are included in the full paper. We observe that after 4 to 5 hours, Quartz (with preprocessing) catches up to QUESO (without preprocessing) on the IBM gate set on 6 of the benchmarks. With the longer timeout, Quartz overall performs slightly better than before on the IBM and Nam gate sets and remains the same on the Rigetti gate set. The notable exception is the comparison against QUESO and Quartz-NoPP on the Nam gate set where QUESO benefits from the longer timeout.

Q2 Summary. **QUESO synthesizes rules for the same size and gate set up to 30x times faster than Quartz. When comparing synthesized rules, QUESO outperforms or matches Quartz on 97% of the benchmarks across all gate sets.**

Q3: Which synthesized rewrite rules are useful?

We explore this question at two levels of granularity: (A) Is there a subset of the rewrite rules that is sufficient for producing optimal circuits? (B) Which classes of rules are useful for optimization? We present results for the representative IBM gate set.

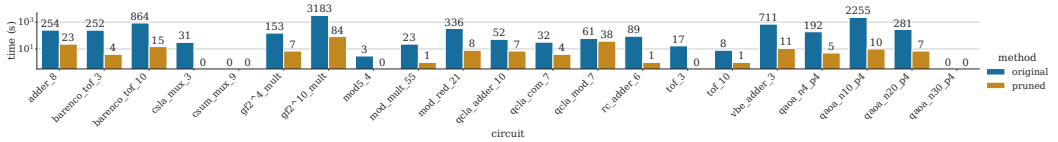


Fig. 13. **Log-scale** comparison of QUESO running time on IBM with pruned (35) vs original (701) rewrite rules.

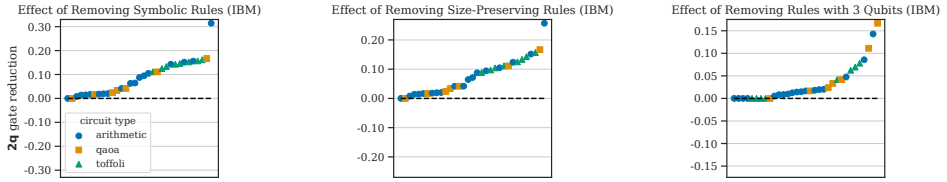


Fig. 14. Effect of different types of rules. Points > 0 indicate that removing the rules is detrimental.

(A) Results. We collect the set of all rewrite rules that result in the best circuit that QUESO can discover within 1 hour of execution time. Across all benchmarks, we observe that a fixed subset of only 35 rules out of 701 rules are used to reach the best solution. This implies that we can run QUESO with a subset of the rules and achieve similar results in a significantly smaller amount of time. We envision, for instance, that QUESO can be *finetuned* on a given class of problems to collect all relevant rules and discard the unnecessary ones.

To understand the time savings, we run QUESO with the *pruned* set of 35 rules and ask: how long does it take for it to reach a circuit of equal cost (in terms of two-qubit gate count) to that found by QUESO running on the full set of rules for 1 hour. Fig. 13 shows the results for a representative subset of the 33 benchmarks. We observe a drastic reduction in runtime. For instance, on the qaoa_n10_p4 benchmark, it takes QUESO 2,255 seconds to converge to the best circuit that can be found within 1 hour, but it takes the pruned version only 10 seconds to arrive at a circuit with the same number of two-qubit gates. Overall, we observe runtime reductions of up to 225x, making QUESO run in a few seconds to a minute on the majority of the benchmarks.

(B) Rule classes. Next, we study the effect of symbolic rules, size-preserving rules, and rules over 3-qubit circuits. Fig. 14 shows the S-curves comparing QUESO versus QUESO without a subset of the rules. The graphs show that removing each type of rule significantly affects the performance—almost all the points are above the dashed line. Most importantly, this shows that being able to synthesize and apply symbolic rules is critical. We also observe that greedily applying only size-reducing rules, results in a worse solution in almost all the benchmarks. Finally, we see a slightly less dramatic effect when removing rules with 3 qubits, i.e., restricting QUESO to rules with up to 2 qubits, but a majority of the benchmarks still rely on these rules.

Q3 Summary. **Restricting QUESO to the small subset of rules used decreases the time to reach the best solution by up to 225x. Symbolic rules significantly contribute to QUESO's performance and the ability to synthesize and apply them is critical.**

8 RELATED WORK

Rewrite-rule synthesis. Most quantum-circuit compilers use hand-crafted optimizations for a given gate set [Aleksandrowicz et al. 2019; Hietala et al. 2021; Nam et al. 2018; Shi et al. 2019; Sivarajah et al. 2020; Smith et al. 2020]. Our approach is most similar to Quartz [Xu et al. 2022a] (and its predecessor [Pointing et al. 2021]) and related work on rewrite-rule synthesis for computation graphs [Jia et al. 2019] and LLVM. QUESO differs from Quartz along two dimensions: expressivity and speed. QUESO can learn symbolic rules, unlike Quartz. These symbolic rules enable optimizations

similar to Nam et al. [2018]'s rotation merging, which Quartz applies as an unverified hand-crafted preprocessing step. Our synthesis phase is much faster due to the application of Schwartz–Zippel as opposed to SMT-based verification. Quartz implements a preprocessing pass to optimize the circuit before beginning to apply rewrite rules using a cost-based backtracking search. In contrast, we rely only on learned rules; as our goal is not superoptimization, we apply rules greedily, allowing our approach to find smaller circuits faster. ZX calculus-based optimizers use graphical rewrite rules [Cowtan et al. 2020; Kissinger and van de Wetering 2019a,b]. However, PyZX's full optimization pass involves a subroutine [PyZX 2023] that only supports circuits with Clifford + T gates and does not support arbitrary rotation, making it rigid and not compatible with existing hardware platforms that execute gates with arbitrary rotation. QUESO is designed to leverage hardware gates to unlock broad optimization opportunities and enable flexibility to adapt to changes in the basis gates.

Verified optimizers. Compilers are hard to get right [Sun et al. 2016], and much progress has been made in building verified classical compilers [Kumar et al. 2014; Leroy 2009]. This includes verified optimizing compilers, using interactive theorem proving [Barthe et al. 2014; Becker et al. 2022; Courant and Leroy 2021; Mullen et al. 2016] and automated techniques like SMT-solving [Lerner et al. 2003; Lopes et al. 2021, 2015]. In the quantum realm, similar efforts have included reversible circuit compilers verified in F* [Amy et al. 2017; Rand et al. 2018], the optimizer VOQC [Hietala et al. 2021], which has been formally verified in Coq, and Giallar [Tao et al. 2022] for verification of Qiskit optimizations. Our work, in contrast to the above, automatically synthesizes probabilistically verified rewrite rules, relying on novel verification insights for this problem domain and a probabilistic data structure. Leveraging PIT for equivalence-checking has been applied in other areas such as program analysis [Gulwani and Necula 2003] and machine learning [Wang et al. 2021].

Circuit resynthesis. There are quantum-circuit optimizations that QUESO cannot discover. The most interesting is that of Cross et al. [2019]. This optimization finds maximal disjoint blocks of gates that operate on a given control and target of a CX in the block. For each two-qubit block, the optimization computes a unitary operation and resynthesizes a subcircuit for the block using either exact techniques [Bullock and Markov 2003; Shende et al. 2004] or approximation. Other similar optimizations include QUEST [Patel et al. 2022], which performs approximate resynthesis of circuits to reduce their CX count, and several exact resynthesis techniques for reducing the CX counts in circuits [Davis et al. 2020; de Brugière et al. 2020; Meuli et al. 2018].

9 CONCLUSIONS AND FUTURE WORK

We have described a technique for automatically generating quantum-circuit optimizers by synthesizing symbolic rewrite rules. Our results demonstrate the remarkable ability of our synthesized optimizers to outperform or rival state-of-the-art optimizers. For future work, we would like to explore (1) learning-based techniques for scheduling rewrite rules to speed up optimization, and (2) enhancements of symbolic rules to capture more sophisticated optimizations like those in Quilc.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers and our shepherd, Joseph Tassarotti, for their insightful feedback. We also thank Justin Hsu and Thomas Reps for their input during the writing process. We are grateful to Martin Diges, Mingkuan Xu, and the CHTC team for their assistance in our experimentation as well as Max Willsey for providing guidance during our exploration of egg.

This work is supported by NSF grants #1652140 and #2212232 and awards from Meta and Amazon. This research is also partially supported by the OVCRGE at the University of Wisconsin–Madison with funding from the Wisconsin Alumni Research Foundation. Lauren Pick is supported by NSF grant #2127309 to the Computing Research Association for the CIFellows Project.

SOFTWARE AVAILABILITY

Our artifact is publicly available on Zenodo [Xu et al. 2023]. It contains the QUESO source code, evaluation infrastructure, and documentation.

REFERENCES

Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, Francisco Jose Cabrera-Hernández, Jorge Carballo-Franquis, Adrian Chen, Chun-Fu Chen, Jerry M. Chow, Antonio D. Córcoles-Gonzales, Abigail J. Cross, Andrew Cross, Juan Cruz-Benito, Chris Culver, Salvador De La Puente González, Enrique De La Torre, Delton Ding, Eugene Dumitrescu, Ivan Duran, Pieter Eendebak, Mark Everitt, Ismael Faro Sertage, Albert Frisch, Andreas Fuhrer, Jay Gambetta, Borja Godoy Gago, Juan Gomez-Mosquera, Donny Greenberg, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Hiroshi Horii, Shaohan Hu, Takashi Imamichi, Toshinari Itoko, Ali Javadi-Abhari, Naoki Kanazawa, Anton Karazeev, Kevin Krsulich, Peng Liu, Yang Luh, Yunho Maeng, Manoel Marques, Francisco Jose Martín-Fernández, Douglas T. McClure, David McKay, Srujan Meesala, Antonio Mezzacapo, Nikolaj Moll, Diego Moreda Rodríguez, Giacomo Nannicini, Paul Nation, Pauline Ollitrault, Lee James O'Riordan, Hanhee Paik, Jesús Pérez, Anna Phan, Marco Pistoia, Viktor Prutyanov, Max Reuter, Julia Rice, Abdón Rodríguez Davila, Raymond Harry Putra Rudy, Mingi Ryu, Ninad Sathaye, Chris Schnabel, Eddie Schoute, Kanav Setia, Yunong Shi, Adenilton Silva, Yukio Siraichi, Seyon Sivarajah, John A. Smolin, Mathias Soeken, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor, Pete Taylour, Kenso Trabing, Matthew Treinish, Wes Turner, Desiree Vogt-Lee, Christophe Vuillot, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, Christopher Wood, Stephen Wood, Stefan Wörner, Ismail Yunus Akhalwaya, and Christa Zoufal. 2019. Qiskit: An Open-source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2562111

Matthew Amy. 2019. Towards Large-scale Functional Verification of Universal Quantum Circuits. *Electronic Proceedings in Theoretical Computer Science* 287 (01 2019), 1–21. https://doi.org/10.4204/EPTCS.287.1

Matthew Amy, Dmitri Maslov, and Michele Mosca. 2014. Polynomial-Time T-Depth Optimization of Clifford+T Circuits Via Matroid Partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33, 10 (2014), 1476–1489. https://doi.org/10.1109/TCAD.2014.2341953

Matthew Amy, Martin Roetteler, and Krysta M. Svore. 2017. Verified Compilation of Space-Efficient Reversible Circuits. In *CAV (2) (Lecture Notes in Computer Science)*, Vol. 10427. Springer, 3–21. https://doi.org/10.1007/978-3-319-63390-9_1

Gilles Barthe, Delphine Demange, and David Pichardie. 2014. Formal Verification of an SSA-Based Middle-End for CompCert. *ACM Trans. Program. Lang. Syst.* 36, 1, Article 4 (mar 2014), 35 pages. https://doi.org/10.1145/2579080

Heiko Becker, Robert Rabe, Eva Darulova, Magnus O. Myreen, Zachary Tatlock, Ramana Kumar, Yong Kiam Tan, and Anthony Fox. 2022. Verified Compilation and Optimization of Floating-Point Programs in CakeML. In *36th European Conference on Object-Oriented Programming (ECOOP 2022) (Leibniz International Proceedings in Informatics (LIPIcs))*, Karim Ali and Jan Vitek (Eds.), Vol. 222. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 1:1–1:28. https://doi.org/10.4230/LIPIcs.ECOOP.2022.1

Stephen S. Bullock and Igor L. Markov. 2003. Arbitrary two-qubit computation in 23 elementary gates. *Phys. Rev. A* 68 (Jul 2003), 012318. Issue 1. https://doi.org/10.1103/PhysRevA.68.012318

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

Nathanaël Courant and Xavier Leroy. 2021. Verified Code Generation for the Polyhedral Model. *Proc. ACM Program. Lang.* 5, POPL, Article 40 (jan 2021), 24 pages. https://doi.org/10.1145/3434321

Alexander Cowtan, Silas Dilkes, Ross Duncan, Will Simmons, and Seyon Sivarajah. 2020. Phase Gadget Synthesis for Shallow Circuits. *Electronic Proceedings in Theoretical Computer Science* 318 (04 2020), 214–229. https://doi.org/10.4204/EPTCS.318.13

Andrew W. Cross, Lev S. Bishop, Sarah Sheldon, Paul D. Nation, and Jay M. Gambetta. 2019. Validating quantum computers using randomized model circuits. *Phys. Rev. A* 100 (Sep 2019), 032328. Issue 3. https://doi.org/10.1103/PhysRevA.100.032328

Marc G. Davis, Ethan Smith, Ana Tudor, Koushik Sen, Irfan Siddiqi, and Costin Iancu. 2020. Towards Optimal Topology Aware Quantum Circuit Synthesis. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. 223–234. https://doi.org/10.1109/QCE49297.2020.00036

Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, Simon Martiel, and Cyril Allouche. 2020. Quantum CNOT Circuits Synthesis for NISQ Architectures Using the Syndrome Decoding Problem. In *Reversible Computation*, Ivan Lanese and Mariusz Rawski (Eds.). Springer International Publishing, Cham, 189–205. https://doi.org/10.1007/978-3-030-52482-1_11

Sumit Gulwani and George C. Necula. 2003. Discovering Affine Equalities Using Random Interpretation. In *Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '03)*. Association for Computing Machinery, New York, NY, USA, 74–84. https://doi.org/10.1145/604131.604138

Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. 2021. A Verified Optimizer for Quantum Circuits. *Proc. ACM Program. Lang.* 5, POPL, Article 37 (jan 2021), 29 pages. https://doi.org/10.1145/3434318

IBM. 2022. IBM Toronto. https://quantum-computing.ibm.com/services/resources?system=ibmq_toronto.

IonQ. 2022a. IonQ Aria. https://ionq.com/posts/july-25-2022-ionq-aria-part-one-practical-performance.

IonQ. 2022b. IonQ Native Gates. https://ionq.com/docs/getting-started-with-native-gates.

Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. 2019. TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 47–62. https://doi.org/10.1145/3341301.3359630

Aleks Kissinger and John van de Wetering. 2019a. Pyzx: Large scale automated diagrammatic reasoning. *arXiv preprint arXiv:1904.04735* (2019).

Aleks Kissinger and John van de Wetering. 2019b. Reducing T-count with the ZX-calculus. *arXiv preprint arXiv:1903.10477* (2019).

Ramana Kumar, Magnus O. Myreen, Michael Norrish, and Scott Owens. 2014. CakeML: A Verified Implementation of ML. *SIGPLAN Not.* 49, 1 (jan 2014), 179–191. https://doi.org/10.1145/2578855.2535841

Sorin Lerner, Todd Millstein, and Craig Chambers. 2003. Automatically Proving the Correctness of Compiler Optimizations. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI '03)*. Association for Computing Machinery, New York, NY, USA, 220–231. https://doi.org/10.1145/781131.781156

Xavier Leroy. 2009. Formal Verification of a Realistic Compiler. *Commun. ACM* 52, 7 (jul 2009), 107–115. https://doi.org/10.1145/1538788.1538814

Nuno P. Lopes, Juneyoung Lee, Chung-Kil Hur, Zhengyang Liu, and John Regehr. 2021. Alive2: bounded translation validation for LLVM. In *PLDI*. ACM, 65–79. https://doi.org/10.1145/3453483.3454030

Nuno P. Lopes, David Menendez, Santosh Nagarakatte, and John Regehr. 2015. Provably correct peephole optimizations with alive. In *PLDI*. ACM, 22–32. https://doi.org/10.1145/2813885.2737965

Giulia Meuli, Mathias Soeken, and Giovanni Micheli. 2018. *SAT-based CNOT, T Quantum Circuit Synthesis: 10th International Conference, RC 2018, Leicester, UK, September 12-14, 2018, Proceedings*. 175–188. https://doi.org/10.1007/978-3-319-99498-7_12

Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized algorithms*. Cambridge university press. https://doi.org/10.1017/CBO9780511814075

Eric Mullen, Daryl Zuniga, Zachary Tatlock, and Dan Grossman. 2016. Verified peephole optimizations for CompCert. In *PLDI*. ACM, 448–461. https://doi.org/10.1145/2908080.2908109

Yunseong Nam, Neil J Ross, Yuan Su, Andrew M Childs, and Dmitri Maslov. 2018. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information* 4, 1 (2018), 1–12. https://doi.org/10.1038/s41534-018-0072-4

Matteo Paltenghi and Michael Pradel. 2022. Bugs in Quantum computing platforms: an empirical study. *Proceedings of the ACM on Programming Languages* 6, OOPSLA1 (2022), 1–27. https://doi.org/10.1145/3527330

Tirthak Patel, Ed Younis, Costin Iancu, Wibe de Jong, and Devesh Tiwari. 2022. QUEST: Systematically Approximating Quantum Circuits for Higher Output Fidelity. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2022)*. Association for Computing Machinery, New York, NY, USA, 514–528. https://doi.org/10.1145/3503222.3507739

Jessica Pointing, Oded Padon, Zhihao Jia, Henry Ma, Auguste Hirth, Jens Palsberg, and Alex Aiken. 2021. Quanto: Optimizing Quantum Circuits with Automatic Generation of Circuit Identities. *CoRR* abs/2111.11387 (2021).

John Preskill. 2018. Quantum computing in the NISQ era and beyond. *Quantum* 2 (2018), 79. https://doi.org/10.22331/q-2018-08-06-79

PyZX. 2023. PyZX Full API documentation. https://pyzx.readthedocs.io/en/latest/api.html#pyzx.optimize.phase_block_optimize.

Google Quantum-AI. 2021. Quantum Computer Datasheet. (Accessed on 11/22/2021).

Robert Rand, Jennifer Paykin, Dong-Ho Lee, and Steve Zdancewic. 2018. ReQWIRE: Reasoning about Reversible Quantum Circuits. In *QPL (EPTCS)*, Vol. 287. 299–312. https://doi.org/10.4204/EPTCS.287.17

Rigetti. 2022. Rigetti Aspen-11. https://www.rigetti.com.

Mark Saffman. 2019. The next step in making arrays of single atoms.

Raimondas Sasnauskas, Yang Chen, Peter Collingbourne, Jeroen Ketema, Gratian Lup, Jubi Taneja, and John Regehr. 2017. Souper: A synthesizing superoptimizer. *arXiv preprint arXiv:1711.04422* (2017).

Vivek V. Shende, Igor L. Markov, and Stephen S. Bullock. 2004. Minimal universal two-qubit controlled-NOT-based circuits. *Phys. Rev. A* 69 (Jun 2004), 062321. Issue 6. https://doi.org/10.1103/PhysRevA.69.062321

Yunong Shi, Runzhou Tao, Xupeng Li, Ali Javadi-Abhari, Andrew W Cross, Frederic T Chong, and Ronghui Gu. 2019. CertiQ: A Mostly-automated Verification of a Realistic Quantum Compiler. *arXiv preprint arXiv:1908.08963* (2019).

Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. 2020. t|ket⟩: a retargetable compiler for NISQ devices. *Quantum Science and Technology* 6, 1 (2020), 014003. https://doi.org/10.1088/2058-9565/ab8e92

Robert S. Smith, Eric C. Peterson, Mark G. Skilbeck, and Erik J. Davis. 2020. An Open-Source, Industrial-Strength Optimizing Compiler for Quantum Programs. *CoRR* abs/2003.13961 (2020). https://doi.org/10.1088/2058-9565/ab9acb

Chengnian Sun, Vu Le, Qirun Zhang, and Zhendong Su. 2016. Toward understanding compiler bugs in GCC and LLVM. In *Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016*, Andreas Zeller and Abhik Roychoudhury (Eds.). ACM, 294–305. https://doi.org/10.1145/2931037.2931074

Runzhou Tao, Yunong Shi, Jianan Yao, Xupeng Li, Ali Javadi-Abhari, Andrew W. Cross, Frederic T. Chong, and Ronghui Gu. 2022. Giallar: Push-Button Verification for the Qiskit Quantum Compiler. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI 2022)*. Association for Computing Machinery, New York, NY, USA, 641–656. https://doi.org/10.1145/3519939.3523431

Haojie Wang, Jidong Zhai, Mingyu Gao, Zixuan Ma, Shizhi Tang, Liyan Zheng, Yuanzhi Li, Kaiyuan Rong, Yuanyong Chen, and Zhihao Jia. 2021. PET: Optimizing Tensor Programs with Partially Equivalent Transformations and Automated Corrections. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, 37–54. https://www.usenix.org/conference/osdi21/presentation/wang

TF Watson, SGJ Philips, Erika Kawakami, DR Ward, Pasquale Scarlino, Menno Veldhorst, DE Savage, MG Lagally, Mark Friesen, SN Coppersmith, et al. 2018. A programmable two-qubit quantum processor in silicon. *nature* 555, 7698 (2018), 633–637. https://doi.org/10.1038/nature25766

Christopher D Wilen, S Abdullah, NA Kurinsky, C Stanford, L Cardani, G d'Imperio, C Tomei, L Faoro, LB Ioffe, CH Liu, et al. 2021. Correlated charge noise and relaxation errors in superconducting qubits. *Nature* 594, 7863 (2021), 369–373. https://doi.org/10.1038/s41586-021-03557-5

Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. 2021. Egg: Fast and extensible equality saturation. *Proceedings of the ACM on Programming Languages* 5, POPL (2021), 1–29. https://doi.org/10.1145/3434304

Amanda Xu, Abtin Molavi, Lauren Pick, Swamit Tannu, and Aws Albarghouthi. 2022b. Synthesizing Quantum-Circuit Optimizers. arXiv:cs.PL/2211.09691

Amanda Xu, Abtin Molavi, Lauren Pick, Swamit Tannu, and Aws Albarghouthi. 2023. Synthesizing Quantum-Circuit Optimizers Artifact (QUESO). https://doi.org/10.5281/zenodo.7809285

Mingkuan Xu, Zikun Li, Oded Padon, Sina Lin, Jessica Pointing, Auguste Hirth, Henry Ma, Jens Palsberg, Alex Aiken, Umut A. Acar, and Zhihao Jia. 2022a. Quartz: Superoptimization of Quantum Circuits. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI 2022)*. Association for Computing Machinery, New York, NY, USA, 625–640. https://doi.org/10.1145/3519939.3523433

Received 2022-11-10; accepted 2023-03-31