Linear and Non-linear Relational Analyses for Quantum Program Optimization

MATTHEW AMY, Simon Fraser University, Canada
JOSEPH LUNDERVILLE, Simon Fraser University, Canada

The phase folding optimization is a circuit optimization used in many quantum compilers as a fast and effective way of reducing the number of high-cost gates in a quantum circuit. However, existing formulations of the optimization rely on an exact, linear algebraic representation of the circuit, restricting the optimization to being performed on straightline quantum circuits or basic blocks in a larger quantum program.

We show that the phase folding optimization can be re-cast as an *affine relation analysis*, which allows the direct application of classical techniques for affine relations to extend phase folding to quantum *programs* with arbitrarily complicated classical control flow including nested loops and procedure calls. Through the lens of relational analysis, we show that the optimization can be powered-up by substituting other classical relational domains, particularly ones for *non-linear* relations which are useful in analyzing circuits involving classical arithmetic. To increase the precision of our analysis and infer non-linear relations from gate sets involving only linear operations — such as Clifford+T — we show that the *sum-over-paths* technique can be used to extract precise symbolic transition relations for straightline circuits. Our experiments show that our methods are able to generate and use non-trivial loop invariants for quantum program optimization, as well as achieve some optimizations of common circuits which were previously attainable only by hand.

CCS Concepts: • **Hardware → Quantum computation**; • **Software and its engineering** → *Compilers*; • **Theory of computation → Program analysis**; *Invariants*.

Additional Key Words and Phrases: Quantum software, compiler optimization, data flow analysis, relational program analysis, invariant generation

ACM Reference Format:
Matthew Amy and Joseph Lunderville. 2025. Linear and Non-linear Relational Analyses for Quantum Program Optimization. *Proc. ACM Program. Lang.* 9, POPL, Article 37 (January 2025), 34 pages. https://doi.org/10.1145/3704873

1 Introduction

The optimization of quantum programs is an increasingly important part of quantum computing research. On the one hand, quantum computers are scaling towards regimes of practical utility, and with it compilation tool chains are increasing in complexity. On the other hand, while the theoretical applications of quantum computers have been well established, the compilation to *fault-tolerant architectures* which is necessary for such applications due to the intrinsically high error rates of quantum processors induces a massive amount of time and space overhead. In order to bring these overheads down to levels where a quantum algorithm can outperform a classical one in practical regimes, researchers have spent much effort optimizing quantum circuits and programs.

A major driver of the overhead is the high cost of *magic state distillation* needed to implement certain gates. In fault-tolerant quantum error correction (FTQEC), logical gates — gates which act on the encoded logical state rather than physical qubits — are typically divided into those that can be implemented directly on the code space via a small number of physical gates, and those

Authors' Contact Information: Matthew Amy, Simon Fraser University, School of Computing Science, Canada; Joseph Lunderville, Simon Fraser University, School of Computing Science, Canada.

which require *gate teleportation* [25] to implement. A classic result in quantum computing [17] states that at least one operation of the latter type is needed to achieve approximate universality. In standard codes, the *Clifford group* generated by the H, S, and CNOT gates can be implemented efficiently on the code space by *transversal* operations [17], *braiding* [23], or *lattice surgery* [36] in modern schemes. The non-Clifford $\mathrm{T} := \mathrm{diag}(1, \omega := e^{\frac{i\pi}{4}})$ gate is typically chosen as the additional gate needed for universality, and is implemented by using magic state distillation to produce a high-fidelity $|\mathrm{T}\rangle := \mathrm{TH}|0\rangle$ state and then teleporting it into a T gate. As a result, the physical footprint of a T gate is often orders of magnitude larger than other gates [36].

A common problem for compilers targeting fault-tolerant quantum computation is hence to reduce the number of T gates in a *Clifford+T* quantum circuit, called its *T-count*, and many methods [5, 6, 16, 27, 35, 40, 46, 59] have been devised to do so. One standard method originating in [5] is colloquially known as *phase folding*, which optimizes circuits by finding pairs of T gates which can be canceled out or otherwise merged into Clifford gates via the identity $\mathrm{T}^2 = \mathrm{S}$. The optimization is crucially *efficient* and *monotone*, in that it operates in polynomial time in the size of the circuit, and is *non-increasing on any meaningful cost metric*. As a fast, effective, and monotone optimization, many existing quantum compilers [3, 28, 30, 34, 49] implement some variant of this algorithm. However, the optimization and its variants rely on the linear-algebraic semantics of a quantum circuit, and hence can only be applied to *circuits*, not *programs*. As quantum tool chains scale up, intermediate representations are necessarily leaving the strictly quantum-centric view and incorporating classical computation directly into compiled code [13], making the integration of such circuit optimizations challenging at best and ineffective at worst.

In this work we develop a phase folding algorithm which applies to quantum *programs* which involve both classical and quantum computation. We do this by re-framing the phase folding algorithm as a relational analysis which computes a sound approximation of the classical transitions in a quantum program via an affine subspace defined over the pre- and post-state. In this way we can directly apply existing techniques for affine relation analysis (ARA) such as Karr's seminal analysis [31]. Moreover, our optimization can handle arbitrarily complicated classical control including nested loops and procedure calls, as it amounts to a process of *summarization*. Framing the phase folding algorithm as an approximation of the *affine* relations between the pre- and post-states in a program further allows us to generalize the optimization to *non-linear* relations. We give a non-linear relational analysis for quantum programs which abstracts the classical transitions as an affine variety over $\mathbb{F}_2$, represented via the reduced Gröbner basis of a polynomial ideal. As most common quantum gates implement affine transitions in the classical state space and generate non-linear behaviour via *interference*, we further develop a method of generating precise non-linear transition formulas for the basic blocks in a control-flow graph via *symbolic path integrals*.

Contributions. In summary we make the following contributions:

- We show that quantum phase folding is effectively an affine relation analysis (Section 4).
- We give the first phase folding algorithm which applies non-trivially to *programs* involving both quantum and classical computation (Section 4).
- We give a novel phase folding algorithm which makes use of *non-linear* relations and is again applicable to general programs. We do so by showing that over $\mathbb{F}_2$, precise polynomial relations are relatively simple to compute via Gröbner basis methods (Section 5).
- We show that (relatively) precise transition relations can be extracted from the symbolic path integral of a basic block and used to increase precision in either domain (Section 6).
- Conceptually, we demonstrate that *classical* program analysis techniques may be productively applied to *classical data in superposition*. This is in contrast to most existing work on quantum program analysis, which treat *quantum data* with *quantum domains*.

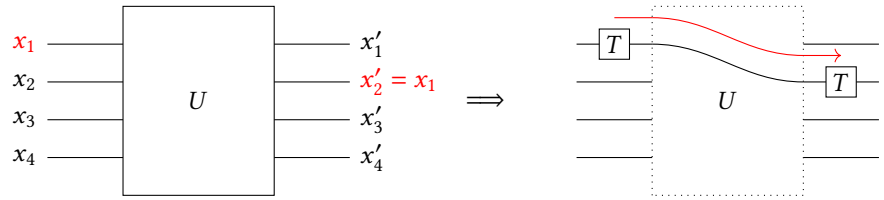


Fig. 1. The relational approach to phase folding. If every classical state $|\vec{x}'\rangle$ in the support of $U|\vec{x}\rangle$ satisfies $x'_j = x_i$, a phase gate on $|x_i\rangle$ can be commuted through U. More generally, a phase conditional on some function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ commutes with U if and only if $f(\vec{x}) = g(\vec{x}')$ for all $\vec{x}'$ in the support of $U|\vec{x}\rangle$.

2 Overview

Figure 1 gives the high-level intuition of our approach. We recall[1] that a *qubit* — a quantum bit — in a pure state is described as a unit vector $|\psi\rangle \in \mathbb{C}^2$. A *quantum gate* U acts on the combined state $|\psi\rangle \in \mathbb{C}^{2^n}$ of n qubits as a unitary (i.e. invertible) linear transformation $|\psi\rangle \mapsto U|\psi\rangle$. A *quantum circuit* is a sequence of quantum gates acting on subsets of the qubits, drawn as a *circuit diagram* where horizontal wires carry qubits from left to right into gates. We fix a basis $\{|\vec{x}\rangle \mid \vec{x} \in \mathbb{F}_2^n := \{0,1\}^n\}$ of $\mathbb{C}^{2^n}$ and call these the *classical states*; a state $|\psi\rangle = \sum_{\vec{x}\in\mathbb{F}_2^n} a_{\vec{x}}|\vec{x}\rangle$ is hence in a *superposition* of the classical states $\{\vec{x} \mid a_{\vec{x}} \neq 0\}$, called the *classical support* of $|\psi\rangle$. We can then describe the *classical* semantics of a quantum gate or circuit as the *classical* states which are in superposition after applying the transformation. *Our methods are based around approximating the classical semantics of a quantum circuit as a set of relations or constraints which hold between $\vec{x}$ and the support of $U|\vec{x}\rangle$.* If certain relations hold, then *phase gates* — gates which are diagonal in the classical basis — can be commuted through the program as in Figure 1 and canceled or merged with other phase gates — a type of *quantum constant propagation.*

To illustrate our methods, consider the circuit in Figure 2a which uses a *repeat-until-success* (RUS) implementation of the gate $\mathrm{V} = (\mathrm{I} + 2i\mathrm{Z})/\sqrt{5} = \mathrm{diag}(\frac{1+2i}{\sqrt{5}}, \frac{1-2i}{\sqrt{5}})$ from [41]. The RUS circuit, corresponding to the boxed loop, implements V by repeatedly initializing two additional (*ancilla*) qubits in the $|0\rangle$ state, performing a computation, then measuring the ancilla qubits to get $a, b \in \mathbb{F}_2$. The resulting transformation on the target qubit after every iteration of the loop is a diagonal gate: I if $ab \neq 00$, and V if $ab = 00$. The (diagonal) T gate before entering the loop can thus cancel out with the $\mathrm{T}^\dagger$ at the end of the loop, as for any diagonal operators U,V we have $UV = VU$.

Achieving this optimization is challenging for existing methods, as the circuit has no finite interpretation as a linear operator. Instead, our method works by generating a (classical) *loop invariant*, namely that for any initial state $|z\rangle$ where $z \in \mathbb{F}_2$, the state at termination has support $\{z\}$ — i.e. the loop is diagonal in the computational basis. To achieve this, we view the classical semantics of a gate as a non-deterministic transition relation $R \subseteq \mathbb{F}_2^n \times \mathbb{F}_2^n$ on the classical states and compute a transition for the entire loop body by composing each relation. The relations for the gates involved are given below, where $|x\rangle \mapsto |x'\rangle$ denotes the relation $\{(x, x') \mid x, x' \in \mathbb{F}_2\}$:

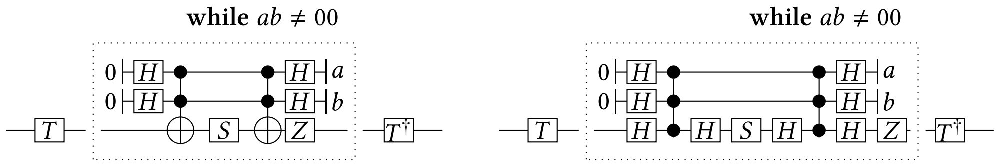
$$-\boxed{H}- = \mathrm{H} : |x\rangle \mapsto |x'\rangle \qquad \text{(CCX)} = \mathrm{TOF} : |x, y, z\rangle \mapsto |x, y, z \oplus (x \wedge y)\rangle$$

$$-\boxed{S}- = \mathrm{S} : |x\rangle \mapsto |x\rangle \qquad -\boxed{Z}- = \mathrm{Z} : |x\rangle \mapsto |x\rangle\,.$$

Composing the relations corresponding to each gate on an initial state $|z\rangle$ and we have

$$|z\rangle \xrightarrow{\text{init}} |0,0,z\rangle \xrightarrow{\mathrm{H}\otimes\mathrm{H}\otimes\mathrm{I}} |x,y,z\rangle \xrightarrow{\mathrm{TOF}} |x,y,z\oplus(x\wedge y)\rangle \xrightarrow{\mathrm{I}\otimes\mathrm{I}\otimes\mathrm{S}} |x,y,z\oplus(x\wedge y)\rangle \xrightarrow{\mathrm{TOF}} |x,y,z\rangle \xrightarrow{\mathrm{H}\otimes\mathrm{H}\otimes\mathrm{Z}} |x',y',z\rangle \xrightarrow{\text{meas}} |z\rangle$$

[1] A full introduction to quantum computation and circuits can be found in Section 3.



(a) A circuit using a repeat-until-success gate.

(b) The same circuit with Toffoli gates expanded as CCZ gates conjugated by H gates.

Fig. 2. Repeat-Until-Success circuits. The box denotes a loop which terminates exactly when the measurement result xy of the first two qubits is 00.

We then take *Kleene closure* of the loop body transition $\text{RUS}_{\text{body}} : |z\rangle \mapsto |z\rangle$, which approximates (in this case, precisely) the composition $\text{RUS}^k_{\text{body}}$ of *any number k of loop iterations* as $|z\rangle \mapsto |z\rangle$. Hence for any number of iterations of the loop RUS_{body}, the state of the qubit in the computational basis is unchanged and so the T gate commutes through and cancels out the $\text{T}^\dagger = \text{T}^{-1}$ gate.

The problem becomes more challenging when the circuit is written in the Clifford+T gate set, or similar gate sets such as Clifford+CCZ which implement classical non-linearity via *interference*. Figure 2b gives an alternative implementation of the RUS circuit over the Clifford+CCZ. The TOF gates have been replaced with the CCZ symbol = CCZ gates conjugated by Hadamard gates, via the circuit equality $\text{TOF} = (\text{I} \otimes \text{I} \otimes \text{H})\text{CCZ}(\text{I} \otimes \text{I} \otimes \text{H})$. The CCZ is diagonal, and hence its classical semantics is given by the identity relation $\text{CCZ} : |x, y, z\rangle = |x, y, z\rangle$. Composing again the relations for each gate, we now find that the loop body has the following classical relation

$$\text{RUS}_{\text{body}} : |z\rangle \mapsto |z'\rangle\,.$$

The relation above over-approximates the precise loop invariant $|z\rangle \mapsto |z\rangle$. Over-approximation is due to the fact that the relation $(\text{I} \otimes \text{I} \otimes \text{H})\text{CCZ}(\text{I} \otimes \text{I} \otimes \text{H}) : |x, y, z\rangle \mapsto |x, y, z'\rangle$ over-approximates the precise semantics of the TOF gate, namely that $z' = z \oplus (x \wedge y) = z \oplus xy$, itself a consequence of the quantum mechanical effect of *interference* which can cause classical transitions to *cancel*.

To recover the precise classical semantics, we use *path integrals* to analyze interference and compute accurate transition relations. Computing the path integral for $(\text{I} \otimes \text{I} \otimes \text{H})\text{CCZ}(\text{I} \otimes \text{I} \otimes \text{H})$ gives the *quantum* transition relation:

$$(\text{I} \otimes \text{I} \otimes \text{H})\text{CCZ}(\text{I} \otimes \text{I} \otimes \text{H}) : |x, y, z\rangle \mapsto \frac{1}{2}\sum_{z'}\left(\sum_{z'' \in \mathbb{F}_2} (-1)^{zz'' \oplus xyz'' \oplus z''z'}\right)|x, y, z'\rangle\,.$$

From the expression above we see that whenever $z' \neq z \oplus xy$ in the outer sum, the *paths* corresponding to the inner sum over z'' have opposite phase (± 1) and hence cancel out, so the circuit satisfies the additional relation $z \oplus xy \oplus z' = 0$. Performing this analysis for the loop in Figure 2b we get the following system of equations, where $t_1, \ldots, t_4$ denote intermediate variables corresponding to the outputs of the 4 H gates on the target qubit:

$$z \oplus xy \oplus t_2 = 0 \qquad t_2 \oplus xy \oplus t_4 = 0 \qquad z' \oplus t_4 = 0$$

Solving the system of polynomial equations above for z' over the input variable z we find that $z' = z$, as was the case for Figure 2a. While simple back substitution suffices in this case, in more general cases the primed output variables may not have a simple solution in terms of the input variables (or indeed, may have no solution at all). We solve the system of polynomial equations by computing a *Gröbner basis* of the ideal generated by the system of equations, and then using

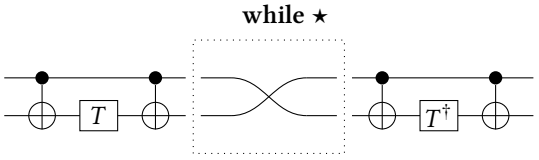


Fig. 3. A circuit with eliminable T gates. The strongest affine loop invariant on the classical support is the relation $x' \oplus y' = x \oplus y$, which implies that the total phase contribution of both T gates is 1.

elimination theory to project out temporary variables. Doing so for the above system of equations produces the polynomial ideal $\langle z \oplus z' \rangle \subseteq \mathbb{F}_2[z, z']$ which in particular implies that $z' = z$.

In the case of the RUS circuit, the loop invariant is simply $z' = z$, but in more exotic cases we can make use of more interesting loop invariants to eliminate of merge phase gates. Consider the circuit in Figure 3 which contains a single non-deterministic while loop which swaps the two qubits. It can be observed that as a function of the classical basis, the loop body either sends $|x, y\rangle \mapsto |x, y\rangle$ or to $|y, x\rangle$ depending on the parity of the number of iterations, and so neither x' nor y' has a solution over the input state. However, it can be observed that the relation $x' \oplus y' = x \oplus y$ holds in either case, and is hence a classical invariant of the loop. Moreover, as this is an *affine* relation we can compute the Kleene closure over a weaker (but computationally more efficient) domain of *affine relations*. This invariant then suffices to eliminate both T gates, as they contribute conjugate phases of ω and $\overline{\omega}$ when $x \oplus y = 1$ and $x' \oplus y' = 1$, respectively, hence the total phase is 1.

3 Preliminaries

We begin by reviewing the basics of quantum computing and Dirac notation, followed by our program model and the phase folding optimization. For a more complete introduction to quantum computation, the reader is directed to [41].

3.1 Quantum Computing

Let $\mathcal{H}$ denote a finite-dimensional Hilbert space with dimension $\dim(\mathcal{H})$. We typically assume $\mathcal{H} = \mathbb{C}^d$ for some fixed dimension d. A *pure* state of a quantum system is a unit vector $|v\rangle \in \mathcal{H}$. The Hermitian adjoint of $|v\rangle$ in $\mathcal{H}^*$, denoted by $\langle v|$, is the linear functional $\langle v| : |u\rangle \mapsto \langle v|u\rangle$ where $\langle v|u\rangle$ denotes the (Hermitian) inner product of $|v\rangle$ and $|u\rangle$. Concretely, the Hermitian adjoint is given by the conjugate transpose, $\langle v| = |v\rangle^\dagger$ and $\langle v|u\rangle = \langle v| \cdot |u\rangle$. The outer product of $|v\rangle$ and $|u\rangle$ is similarly written $|v\rangle\langle u|$. Given two vectors $|v\rangle \in \mathcal{H}_1$, $|u\rangle \in \mathcal{H}_2$ in Hilbert spaces $\mathcal{H}_1, \mathcal{H}_2$, we write $|v\rangle \otimes |u\rangle \in \mathcal{H}_1 \otimes \mathcal{H}_2$ for their combined state, where $\otimes$ is the tensor product. If $|\psi\rangle \in \mathcal{H}_1 \otimes \mathcal{H}_2$ can not be written as a tensor product of states of $\mathcal{H}_1$ and $\mathcal{H}_2$ then $|\psi\rangle$ is said to be *entangled* or *non-separable*. Recall that $\dim(\mathcal{H}_1 \otimes \mathcal{H}_2) = \dim(\mathcal{H}_1)\dim(\mathcal{H}_2)$, and given orthonormal bases $\{|e_i\rangle\}$, $\{|f_j\rangle\}$ of $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively, $\{|e_i\rangle \otimes |f_j\rangle\}$ forms an orthonormal basis of $\mathcal{H}_1 \otimes \mathcal{H}_2$. We often write $|v, u\rangle$ for the tensor product $|v\rangle \otimes |u\rangle$ when the meaning is clear. Given a set of quantum variables Q, corresponding to distinct subsystems q with local Hilbert space $\mathcal{H}_q$, we denote the Hilbert space of the combined system as $\mathcal{H}_Q := \bigotimes_{q \in Q} \mathcal{H}_q$. Tensor factors may be rearranged at will, and so we will sometimes use the notation $|\psi\rangle_{\vec{q}} \otimes |\varphi\rangle_{Q - \vec{q}}$ to denote a separable state where $|\psi\rangle$ is the state of the qubits listed in $\vec{q} = q_1 \cdots q_k$ and $|\varphi\rangle$ is the state of the remaining qubits.

The Hilbert space associated to a 2-dimensional quantum state — called a *qubit* — is $\mathbb{C}^2$ and we denote the standard or *computational* basis of $\mathbb{C}^2$ by $\{|0\rangle, |1\rangle\}$. The n-fold tensor product of $\mathbb{C}^2$ is $\mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2 \simeq \mathbb{C}^{2^n}$ with computational basis $\{|\vec{x}\rangle \mid \vec{x} \in \mathbb{F}_2^n\}$ where $|x_1 x_2\rangle = |x_1\rangle \otimes |x_2\rangle$ and $\mathbb{F}_2 = (\{0, 1\}, \oplus, \cdot)$ is the 2-element finite field. Written over the computational basis, a pure

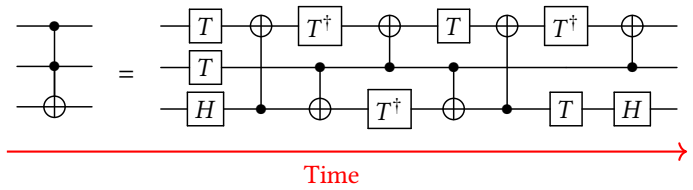


Fig. 4. An example of a quantum circuit, implementing the Toffoli gate TOF : $|x_1, x_2, x_3\rangle \mapsto |x_1, x_2, x_3 \oplus x_1x_2\rangle$

$$\text{X} = \boxed{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \text{Z} = \boxed{Y} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{R}_Z(\theta) = \boxed{R_Z(\theta)} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$$

$$\text{H} = \boxed{H} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \text{S} = \boxed{S} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \quad \text{T} = \boxed{T} = \begin{bmatrix} 1 & 0 \\ 0 & \omega := e^{i\frac{\pi}{4}} \end{bmatrix}$$

$$\text{CNOT} = = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \text{SWAP} = = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fig. 5. Standard gates and their circuit notation.

state of n qubits $|v\rangle = \sum_{\vec{x}\in\mathbb{F}_2^n} \alpha_{\vec{x}} |\vec{x}\rangle$ is said to be in a *superposition* of the classical states $\vec{x} \in \mathbb{F}_2^n$ for which $\alpha_{\vec{x}} \neq 0$. We call this set the *classical support* or just support of a pure state $|v\rangle$, denoted $\text{supp}(|v\rangle)$. Given the pure state $|v\rangle$ above, *measuring* $|v\rangle$ in the computational basis projects the state down to some $|\vec{y}\rangle \in \text{supp}(|v\rangle)$ with probability $|\alpha_{\vec{y}}|^2$. More generally we may measure just a single qubit or subset of qubits — known as a *partial measurement* — which sends a pure state $\sum_{\vec{y}} \alpha_{\vec{y}} |\vec{y}\rangle_{\vec{q}} \otimes |\psi_{\vec{y}}\rangle_{Q-\vec{q}}$ to the pure state $|\vec{y}\rangle_{\vec{q}} \otimes |\psi_{\vec{y}}\rangle_{Q-\vec{q}}$ with probability $|\alpha_{\vec{y}}|^2$.

3.2 Quantum Circuits

A *quantum gate* is a unitary operator U on a Hilbert space $\mathcal{H}$. By unitary we mean that its inverse is equal to its Hermitian adjoint $U^\dagger$, i.e. $UU^\dagger = U^\dagger U = I$. A quantum gate U sends a pure state $|v\rangle$ to the pure state $U|v\rangle$. If $U : \mathcal{H}_1 \to \mathcal{H}_1$ and $V : \mathcal{H}_2 \to \mathcal{H}_2$ are unitary gates, then their tensor product $U \otimes V : \mathcal{H}_1 \otimes \mathcal{H}_2 \to \mathcal{H}_1 \otimes \mathcal{H}_2$ is likewise unitary. A *quantum circuit* is a well-formed term over the signature $(\mathcal{G}, \cdot, \otimes, \dagger)$ where $\mathcal{G}$ is a finite set of gates. We use $U_{\vec{q}}$ to denote a unitary on $\mathcal{H}_Q$ where U is applied to the sub-systems $\vec{q}$ and the identity gate I is applied to the remaining sub-systems. Circuits are drawn graphically as in Figure 4, with time flowing left to right and individual wires corresponding to qubits. Vertical composition corresponds to the tensor product $\otimes$, while horizontal composition corresponds to functional composition $(\cdot)$.

Standard gates are shown in Figure 5. The *Clifford+T* gate set is defined as $\{\text{H}, \text{CNOT}, \text{S}, \text{T}\}$ where $\text{S} = \text{T}^2$ is included as an explicit generator since, being Clifford, it is typically orders of magnitude less expensive than the T gate. It is often illuminating to study quantum gates by their actions on *classical* states, which suffices due to the linearity of quantum mechanics. The gates of Figure 5 expressed as functions of classical states are shown below. In this view, it is clear that X and CNOT both perform *classical* functions, while the Z, S and T operate in the (orthogonal) phase space, and the H is a form of *quantum branching* gate, sending a classical state $|x\rangle$ to both $|0\rangle$ and $|1\rangle$ with equal probability but varying phase. By orthogonal here we mean that computations in the

state space (or Z-basis in quantum mechanical terms) don't affect the phase, and vice versa with computations in the phase space or X-basis.

$$\textsc{x} : |x\rangle \mapsto |1 \oplus x\rangle \qquad \textsc{z} : |x\rangle \mapsto (-1)^x |x\rangle \qquad \textsc{r}_Z(\theta) : |x\rangle \mapsto e^{i\theta x} |x\rangle$$

$$\textsc{h} : |x\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{y \in \mathbb{F}_2} (-1)^{xy} |y\rangle \qquad \textsc{s} : |x\rangle \mapsto i^x |x\rangle \qquad \textsc{t} : |x\rangle \mapsto \omega^x |x\rangle$$

$$\textsc{cnot} : |x, y\rangle \mapsto |x, x \oplus y\rangle \qquad \textsc{swap} : |x, y\rangle \mapsto |y, x\rangle$$

The interpretation of quantum mechanical processes as *classical* processes happening in superposition is central to Feynman's *path integral* formulation of quantum mechanics [22], and the above expressions can be thought of as *symbolic path integrals*.

3.3 Programming Model

We adopt the standard QRAM model of quantum computation, where *data* can be quantum or classical, but *control* is strictly classical. We illustrate our techniques on a non-deterministic version of the imperative quantum WHILE language [20, 57] with procedures, the syntax of which is presented below:

$$T \in \textbf{QWhile} ::= \textbf{skip} \mid q := |0\rangle \mid U\vec{q} \mid \textbf{meas}\ q \mid \textbf{call}\ p(\vec{q}) \mid T_1;\ T_2$$
$$\mid \textbf{if}\ \star\ \textbf{then}\ T_1\ \textbf{else}\ T_2 \mid \textbf{while}\ \star\ \textbf{do}\ T$$

We assume a fixed set of quantum variables $\mathcal{Q}$ and a set of primitive gates $\mathcal{G}$. Variables $q \in \mathcal{Q}$ denote qubit identifiers, while $U \in \mathcal{G}$ denote primitive gates. Note that the syntax corresponds precisely to the syntax of regular expressions over a grammar consisting of reset instructions, unitary applications, measurements in the computational basis, and procedure calls.

We are interested in *relational* properties of quantum WHILE programs — that is, properties which relate the post-state of a program T to the pre-state. Such analyses often operate by computing a *summary* or *abstract transformer* $[\![T]\!]^\sharp : \mathcal{D} \to \mathcal{D}$ for the program over an abstract domain $\mathcal{D}$ of relevant properties. Tarjan [52] introduced a compositional approach to these types of analyses, sometimes called *algebraic program analysis*, in which summarization proceeds by first solving the *path expression problem* — representing a program's control-flow graph as a regular expression — and then reinterpreting the regular expression over a suitable algebraic domain of abstract transformers, notably one equipped with the regular algebra notions of *choice* (denoted + or $\sqcup$), *composition* ($\cdot$ or $\fatsemi$), and *iteration* ($\star$). Our methods are based on this algebraic style of program analysis, and in particular can be applied more to any control-flow graph with quantum instructions by first mapping it to a path expression via Tarjan's algorithm [51].

To this end, we define the semantics of a quantum WHILE program in terms of control-flow paths. We define a set of *actions* Σ as either the application of a unitary or the post-selection of a variable q on a classical value x, with a special n-op action for convenience:

$$\Sigma ::= \textbf{skip} \mid U\vec{q} \mid \textbf{assume}\ P_q^x$$

A *control-flow path* $\pi \in \Sigma^*$ is a sequence of actions. We denote the sequential composition of paths by $\pi \fatsemi \pi'$ and extend $\fatsemi$ to sets of paths. We define the semantics $[\![\pi]\!] : \mathcal{H}_Q \to \mathcal{H}_Q$ of a control-flow path π as a composition of linear operators corresponding to the individual actions, where for instance $[\![U\vec{q}]\!] = U_{\vec{q}}$ and $P^x = |x\rangle\langle x|$ is the projection onto the basis state $|x\rangle$. We say a path π is *feasible* if it is non-zero as a linear operator — that is, there exists $|v\rangle \in \mathcal{H}_Q$ such that $[\![\pi]\!]\,|v\rangle \neq 0$. An example of an infeasible path is one which first projects on to the $|0\rangle$ state of a qubit, then the $|1\rangle$ state:

$$[\![\textbf{assume}\ P_q^0;\ \textbf{assume}\ P_q^1]\!] = P_q^1 P_q^0 = 0.$$

$$\mathcal{T}[\![\textbf{skip}]\!] = \{\textbf{skip}\} \qquad \mathcal{T}[\![\textbf{call } p(\vec{q})]\!] = \mathcal{T}[\![p(\vec{q})]\!]$$
$$\mathcal{T}[\![q := |0\rangle]\!] = \{\textbf{assume } P^0_q\} \cup \{\textbf{assume } P^1_q;\ Xq\} \qquad \mathcal{T}[\![T_1;\ T_2]\!] = \mathcal{T}[\![T_1]\!] \fatsemi \mathcal{T}[\![T_2]\!]$$
$$\mathcal{T}[\![U\vec{q}]\!] = \{U\vec{q}\} \qquad \mathcal{T}[\![\textbf{if } \star \textbf{ then } T_1 \textbf{ else } T_2]\!] = \mathcal{T}[\![T_1]\!] \cup \mathcal{T}[\![T_2]\!]$$
$$\mathcal{T}[\![\textbf{meas } q]\!] = \{\textbf{assume } P^0_q\} \cup \{\textbf{assume } P^1_q\} \qquad \mathcal{T}[\![\textbf{while } \star \textbf{ do } T]\!] = \cup_{k\geq 0}\mathcal{T}[\![T]\!]^k$$

Fig. 6. Path collecting semantics for the non-deterministic quantum WHILE language.

Note that with this interpretation, a path might not send a unit vector to another unit vector (hence, pure state). As we are only interested in the *support* of a path, this definition suffices for our purposes.

Figure 6 defines a collecting semantics for the quantum WHILE language in terms of paths. Paths correspond to control-flow paths over two different types of classical non-determinism: classical branching corresponding to if and while statements, and branching due to measurement outcomes of quantum states. Measurements are modeled as a non-deterministic choice between the projector $P^0 = |0\rangle\langle 0|$ or the projector $P^1 = |1\rangle\langle 1|$. Reset statements are modeled as a measurement followed by an X correction in the event of measurement outcome 1. The path semantics induces a collecting semantics $[\![T]\!] : \mathcal{P}(\mathcal{H}_Q) \to \mathcal{P}(\mathcal{H}_Q)$ in the obvious way. It can be observed that up to normalization, $|u\rangle \in [\![T]\!]\{|v\rangle\}$ if and only if it results from some sequence of measurements and (non-deterministic) classical branches with non-zero probability. In particular, it can be observed that measuring a qubit which has been reset to $|0\rangle$ has no effect, as shown below where $S \in \mathcal{P}(\mathcal{H}_Q)$:

$$[\![q := |0\rangle;\ \textbf{meas } q]\!]S = [\![q := |0\rangle]\!]S = \{|0\rangle_q \otimes |\psi\rangle_{Q-\{q\}} \mid |x\rangle_q \otimes |\psi\rangle_{Q-\{q\}} \in S\}.$$

Example 1. Consider the program $Hq;\ q := |0\rangle$ which applies a Hadamard gate to qubit q then resets q to $|0\rangle$. Up to normalization we have

$$[\![Hq;\ q := |0\rangle]\!]\{|0\rangle\} = \{|0\rangle\langle 0|\,\mathrm{H}\,|0\rangle\} \cup \{\mathrm{X}\,|1\rangle\langle 1|\,\mathrm{H}\,|0\rangle\} = \{|0\rangle\}$$

We extend $\mathrm{supp}(\cdot)$ to sets of quantum states in the obvious way. Given a set $s \in \mathcal{P}(C_Q)$ where $C_Q := \mathbb{F}_2^{|Q|}$ is the set of classical states on Q we can conversely take the linear span of quantum states $\{|\vec{x}\rangle \mid \vec{x} \in s\}$ as an element of $\mathcal{P}(\mathcal{H}_Q)$. We denote this operator simply as $\mathrm{span}(s) \in \mathcal{P}(\mathcal{H}_Q)$. Note that $\mathrm{supp}(\cdot)$ and $\mathrm{span}(\cdot)$ form a Galois connection between $(\mathcal{P}(\mathcal{H}_Q), \subseteq)$ and $(\mathcal{P}(C_Q), \subseteq)$:

$$(\mathcal{P}(\mathcal{H}_Q), \subseteq) \underset{\mathrm{supp}}{\overset{\mathrm{span}}{\leftrightarrows}} (\mathcal{P}(C_Q), \subseteq)$$

A conceptual contribution of this paper is the fact that we may productively analyze a quantum program which includes both quantum and classical data flow (i.e. in and out of superposition) as a *purely classical program* by abstracting its classical semantics $C[\![\cdot]\!] : \mathcal{P}(C_Q) \to \mathcal{P}(C_Q)$.

3.4 Quantum Phase Folding

The *phase folding* algorithm originated in [5] from the *phase polynomial* representation of CNOT-dihedral operators. It was shown that any circuit over the CNOT-dihedral gate set $\{\mathrm{CNOT}, \mathrm{X}, \mathrm{R}_Z(\theta) \mid \theta \in \mathbb{R}\}$ implements a unitary transformation which can be described as

$$U : |\vec{x}\rangle \mapsto e^{2\pi i f(\vec{x})}\,|A\vec{x} + \vec{c}\rangle$$

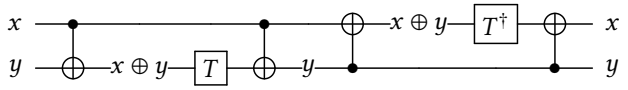
where $(A, \vec{c}) : \vec{x} \mapsto A\vec{x} + \vec{c}$ is an affine transformation over $\mathbb{F}_2$, and $f(\vec{x}) = \sum_i a_i f_i(\vec{x}) \in \mathbb{R}$ where each $f_i : \mathbb{F}_2^n \to \mathbb{F}_2$ is a *linear* function of $\vec{x}$. Note that a linear functional $f_i : \mathbb{F}_2^n \to \mathbb{F}_2$ is an element of the *dual space* $(\mathbb{F}_2^n)^*$ and can hence be represented as a (row) vector. We use the two notions

interchangeably. We call the linear functionals $\{f_i\}_i \subseteq (\mathbb{F}_2^n)^*$ for which $a_i \neq 0$ the *support* of f. The utility of the phase polynomial representation for circuit optimization relies on two facts [5]:

(1) for any CNOT-dihedral circuit U, the canonical phase polynomial of U can be efficiently computed and its support has size at most the number of R_Z gates in the circuit, and
(2) any unitary $U : |\vec{x}\rangle \mapsto e^{2\pi i f(\vec{x})} |A\vec{x} + \vec{c}\rangle$ can be efficiently implemented with exactly $|\text{supp}(f)|$ R_Z gates, with parameters given by the Fourier coefficients a_i.

Letting $\tau(U)$ denote the number of R_Z gates in U, the two facts above give a poly-time method of producing from U a circuit U' where $\tau(U') \leq \tau(U)$ – notably, by computing U's phase polynomial representation, then synthesizing a new circuit with at most $\text{supp}(f) \leq \tau(U)$ R_Z gates.

Example 2. Consider the circuit, for which the intermediate states of the circuit as a function of an initial classical state $|x, y\rangle$ have been annotated, below:



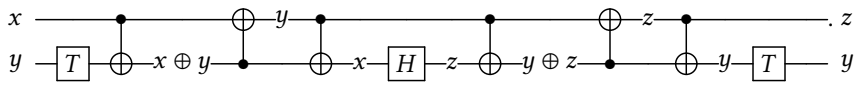
The effect of the first CNOT is to map $|x, y\rangle$ to $|x, x \oplus y\rangle$, at which point the T then applies a phase of ω conditional on the sum $x \oplus y$. After the third CNOT the state is now $|x \oplus y, x\rangle$ at which point the $T^\dagger$ gate applies a phase of $\overline{\omega}$ conditional on the sum $x \oplus y$. It can be readily observed that for any $x, y \in \mathbb{F}_2$ the total phase accumulated by the circuit is 1, as when $x \oplus y = 0$ no phase is accumulated, and when $x \oplus y = 1$ the phase accumulated is $\omega\overline{\omega} = 1$.

The phase polynomial based optimization was extended in [5] to a universal gate set by using path integrals. In particular, a circuit over the universal gate set $\{\text{H}, \text{CNOT}, \text{X}, R_Z(\theta)\}$ can be represented as the mapping

$$U : |\vec{x}\rangle \mapsto \sum_{\vec{y} \in \mathbb{F}_2^k} e^{2\pi i f(\vec{x}, \vec{y})} (-1)^{Q(\vec{x}, \vec{y})} |A(\vec{x}, \vec{y}) + \vec{c}\rangle$$

where f and A are as before, and $Q : \mathbb{F}_2^n \times \mathbb{F}_2^k \to \mathbb{F}_2$ is pure quadratic. While re-synthesis can be performed by splitting the above into CNOT-dihedral sub-circuits, often re-synthesis is not only a bottleneck in efficiency, but may actually have the undesirable affect of *increasing* the circuit cost in some other metric. For this reason, subsequent algorithms [2, 40] dropped re-synthesis in favour of *merging* existing R_Z gates provided they contribute to the same term of f. This also allowed R_Z gates with indeterminate parameters to be merged, as $R_Z(\theta)R_Z(\theta') = R_Z(\theta + \theta')$.

Example 3. Consider the circuit below with intermediate states annotated again:



The circuit implements the transformation $|x, y\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{z \in \mathbb{F}_2} \omega^{2y} (-1)^{xz} |z, y\rangle$ where z denotes to the branch taken (in superposition) by the Hadamard gate. Here we can see that the T and $T^\dagger$ gates both contribute phases of ω^y, giving a combined phase of $\omega\omega^{2y} = i^y$. Hence we can replace the two T gates with a single S gate at either location.

In [2], one of the authors extended the optimization to include *uninterpreted gates* — gate symbols where a concrete semantics exists but is not known to the compiler. Such gates can be used to model gates from non-native gate sets, or *opaque* gates and library calls as in the openQASM language [13]. Intuitively, given an uninterpreted gate U on n qubits, U can be assumed to map a classical state $|x_1 x_2 \cdots x_n\rangle$ to an arbitrary classical state $|x'_1 x'_2 \cdots x'_n\rangle$. It was shown that this interpretation is sound with respect to merging R_Z gates, as any rotation conditional on the

primed outputs can't be commuted back through the uninterpreted gate. On the other hand, $(U \otimes I)\,|x_1 x_2 \cdots x_n, y\rangle = |x_1' x_2' \cdots x_n', y\rangle$ and so gates which involve qubits not modified by U may commute and merge freely.

Example 4. The optimization in Example 3 can be observed to be valid even if the Hadamard gate is replaced with an arbitrary (i.e. uninterpreted) gate U, as regardless of the action of U both T gates act on the state $|y\rangle$. While the optimization of Example 3 is achievable by previous phase folding and related techniques [5, 35, 40, 59] which rely on *exact* representations of the circuit semantics, *these techniques are generally insufficient for the above optimization.*

In the context of program analysis, the mapping $U : |x_1 x_2 \cdots x_n\rangle \mapsto |x_1' x_2' \cdots x_n'\rangle$ asserts that *no relations hold* between an input state and output state. Likewise, the CNOT mapping $\text{CNOT} : |x, y\rangle \mapsto |x, y \oplus x\rangle$ asserts that the affine relations $x' = x$ and $y' = x \oplus y$ hold. As the output of a quantum gate may in fact be a *superposition* of classical states $\vec{x}'$, a relation on the pre- and post-state holds if it holds whenever $\langle \vec{x}' |\, U\, |\vec{x}\rangle \neq 0$. This observation serves as the basis of our work.

4 A Relational Approach to Phase Folding

In this section we formulate the phase folding optimization as a relational analysis computing (an approximation of) the affine relations between the classical inputs and outputs of a quantum circuit. We then use existing techniques [18, 31] for affine relational analyses to extend this to non-deterministic quantum WHILE programs.

4.1 From Phase Folding to Subspaces

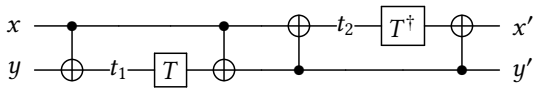
In the phase folding algorithm, the (classical) support of the circuit $|\vec{x}'\rangle$ at any given point in the circuit is described as an affine function of the input variables $\vec{x}$ and intermediate variables $\vec{y}$. In particular, $x_i' = A_i[\vec{x}, \vec{y}]^T + c_i$. A $\text{R}_Z(\theta)$ gate at this location then applies a phase of $e^{i\theta x_i'} = e^{i\theta(A_i[\vec{x},\vec{y}]^T + c_i)}$, i.e. a phase of $e^{i\theta}$ conditional on $f_i(\vec{x}, \vec{y}) = A_i[\vec{x}, \vec{y}]^T + c_i = 1$. Another phase gate which applies a phase conditional on $x_j' = A_j[\vec{x}, \vec{y}]^T + c_j = 1$ contributes to the same term of the phase if and only if $A_i[\vec{x}, \vec{y}]^T = A_j[\vec{x}, \vec{y}]^T$ for all $\vec{x}, \vec{y}$. Note here that the affine factors of x_i' and x_j' may differ, in which case merging the phase gates results in an unobservable global phase.

Relations of the above form correspond [31] to an *affine subspace* over the variables (primed, unprimed, and in our case, intermediate), in the sense that the set of solutions $[\vec{x}', \vec{x}, \vec{y}]^T \in \mathbb{F}_2^{2n+k}$ to the relations $x_i' = A_i[\vec{x}, \vec{y}]^T + c_i$ form an affine subspace of $\mathbb{F}_2^{2n+k}$, where n is the number of program variables and k is the number of intermediate variables. This affine subspace can be represented as the kernel of a *constraint matrix* $T \in \mathbb{F}_2^{n \times 2n+k+1}$, where the rows of T encode the relations or *constraints* $f_i(\vec{x}', \vec{x}, \vec{y}) = c_i$ for linear functions f_i. In particular, the smallest affine subspace satisfying the relations $x_i' = A_i[\vec{x}, \vec{y}]^T + c_i$ can be encoded as the constraint matrix $T = \left[\begin{array}{c|c|c} I & A & \vec{c} \end{array}\right]$, noting that $T[\vec{x}', \vec{x}, \vec{y}, 1]^T = 0$ if and only if $x_i' = A_i[\vec{x}, \vec{y}]^T + c_i$ for every i. More generally, given a set of affine constraints $\{f_i\}_i^k$ (i.e. affine functions), we denote the smallest affine subspace satisfying each f_i by $\langle f_1, f_2, \ldots, f_k \rangle$ and encode it as the kernel of the constraint matrix with rows f_i. As $\mathbb{F}_2$ is a field, this subspace can further be uniquely represented by reducing the constraint matrix to reduced row echelon form.

Given two phases conditional on *linear* functions $f_i, f_j \in (\mathbb{F}_2^{2n+k})^*$ of the primed, unprimed, and intermediate variables, f_i and f_j correspond to row vectors over $\mathbb{F}_2^{2n+k}$ and $f_i(\vec{x}', \vec{x}, \vec{y}) = f_j(\vec{x}', \vec{x}, \vec{y})$ for all $\vec{z} \in \ker T$ if and only if as row vectors, $f_i + f_j$ is in the row space of T up to an (irrelevant) affine factor. Moreover, the restriction of f_i and f_j to $\ker T$ can be uniquely canonicalized by fully row reducing them with respect to T. Hence, to merge all phases conditional on linear functions

over $\vec{x}'$, $\vec{x}$, and $\vec{y}$, we need only row reduce each one with respect to T and collect phases on identical conditions.

Example 5. Consider the circuit below, reproduced from Example 2, with intermediate variables t_1, t_2 at distinguished points.



The non-zero transitions of this circuit lie in the subspace $\langle x' = x, y' = y, t_1 = x \oplus y, t_2 = x \oplus y\rangle$ which we can encode as the constraint matrix on the left below, reduced to row echelon form on the right:

$$\begin{array}{l|cc|cc|cc|c} \text{constraint} & x' & y' & x & y & t_1 & t_2 & c \\ x' = x & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ y' = y & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ t_1 = x \oplus y & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ t_2 = x \oplus y & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{array} \longrightarrow \begin{array}{l|cc|cc|cc|c} \text{constraint} & x' & y' & x & y & t_1 & t_2 & c \\ x' = y \oplus t_2 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ y' = y & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ x = y \oplus t_2 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ t_1 = t_2 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}$$

The two phases of ω and $\overline{\omega}$ are conditional on t_1 and t_2 respectively, corresponding to the linear functionals

$$f = \begin{array}{c|cc|cc|cc} & x' & y' & x & y & t_1 & t_2 \\ & 0 & 0 & 0 & 0 & 1 & 0 \end{array} \quad \text{and} \quad g = \begin{array}{c|cc|cc|cc} & x' & y' & x & y & t_1 & t_2 \\ & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

Row reducing both modulo (the linear part of) the reduced constraint matrix above yields the same linear functional (t_2), hence both phases contribute to the same term and can be canceled.

4.2 Phase Folding as an Affine Relation Analysis

The previous observations are not new in either the classical or in the quantum context, but the shift in perspective allows one to re-frame the phase-folding optimization as computing an affine subspace $\mathcal{S} \subseteq \mathbb{F}_2^{2n}$ corresponding to the classical transitions of a circuit, for which many domains have previously been devised [18, 31, 33, 39]. Before discussing concrete domains and their use in the phase folding optimization, we first formalize phase folding as an affine relation analysis.

Using the terminology of [18], we say that $\mathcal{S}[\vec{X}';\ \vec{X};\ \vec{Y}] \subseteq \mathbb{F}_2^{2n+k}$ is an affine subspace in three *vocabularies* — n primed variables $\vec{X}'$ representing the post-state, n unprimed variables $\vec{X}$ representing the pre-state, and k intermediate variables $\vec{Y}$. The definition below formalizes the notion of such a subspace as a sound abstraction of the classical transitions of a control-flow path.

Definition 6 (sound & precise)**.** We say that an affine subspace $\mathcal{S}[\vec{X}';\ \vec{X};\ \vec{Y}]$ is a *sound abstraction* of a control-flow path $\pi \in \Sigma^*$ if for any $\vec{x}', \vec{x} \in \mathbb{F}_2^n$

$$\vec{x}' \in \mathrm{supp}(\llbracket \pi \rrbracket |\vec{x}\rangle) \implies \exists \vec{y}.(\vec{x}', \vec{x}, \vec{y}) \in \mathcal{S}$$

We say that $\mathcal{S}$ is a *precise* abstraction if the reverse implication also holds.

The condition above that $\exists \vec{y}.(\vec{x}', \vec{x}, \vec{y}) \in \mathcal{S}$ states that $(\vec{x}', \vec{x})$ is contained in the affine subspace $\mathcal{S}'$ of $\mathbb{F}_2^n \times \mathbb{F}_2^n$ with the coordinates corresponding to $\vec{Y}$ projected out. We denote this subspace $\exists \vec{Y}.\mathcal{S}$ and observe that $\mathcal{S}[\vec{X}';\ \vec{X};\ \vec{Y}]$ is a sound (resp. precise) abstraction of π if and only if $\exists \vec{Y}.\mathcal{S}$ is. We denote a subspace in two vocabularies $\vec{X}', \vec{X}$ — for instance, a three-vocabulary subspace with intermediates projected out — by $\mathcal{S}[\vec{X}';\ \vec{X}]$.

Intuitively, a subspace $\mathcal{S}[\vec{X}';\ \vec{X}]$ which soundly approximates a path π over-approximates the set of non-zero transitions $\langle \vec{x}| \llbracket \pi \rrbracket |\vec{x}\rangle$ and can be viewed as a *state transition formula* [32]. Note

that if π is infeasible, then $\mathrm{supp}(\llbracket \pi \rrbracket |\vec{x}\rangle) = \emptyset$ for any $\vec{x}$, hence the special empty subset $\bot = \emptyset$ is a sound abstraction of π. We can compose two state transitions $\mathcal{S}[\vec{X}';\ \vec{X}], \mathcal{S}'[\vec{X}';\ \vec{X}]$ by taking their relational composition

$$(\vec{x}', \vec{x}) \in \mathcal{S} \fatsemi \mathcal{S}' \iff \exists \vec{x}'' \in \mathbb{F}_2^n . (\vec{x}'', \vec{x}) \in \mathcal{S} \wedge (\vec{x}', \vec{x}'') \in \mathcal{S}'$$

which is again an affine subspace corresponding to the intersection of $\mathcal{S}[\vec{X}'';\ \vec{X}]$ and $\mathcal{S}'[\vec{X}';\ \vec{X}'']$ taken as subspaces of the three-vocabulary space $\mathbb{F}_2^{3n}$ followed by a projection of $\vec{X}''$. We denote the projection of $\vec{X}''$ by $\exists \vec{X}''.\mathcal{S}$, and define

$$\mathcal{S} \fatsemi \mathcal{S}' := \exists X''.\mathcal{S}[\vec{X}'';\ \vec{X}] \cap \mathcal{S}'[\vec{X}';\ \vec{X}''].$$

Proposition 7. *If $\mathcal{S}$ and $\mathcal{S}'$ are sound abstractions of π and π', then $\mathcal{S} \fatsemi \mathcal{S}'$ is sound for $\pi \fatsemi \pi'$.*

Proof. Follows from linearity of $\llbracket \pi \fatsemi \pi' \rrbracket = \llbracket \pi' \rrbracket \llbracket \pi \rrbracket$. □

Remark 8. In contrast to classical contexts, composition is *not precise* due to the effects of interference. Notably, the only sound abstraction of H is $\top = \mathbb{F}_2^2$ since we have non-zero transitions between every $|x\rangle$ and $|x'\rangle$. Clearly $\top \fatsemi \top = \top$. On the other hand, $\vec{x}' \in \mathrm{supp}(\llbracket \mathrm{HH} \rrbracket |\vec{x}\rangle) \implies x' = x$, and so a sound abstraction of HH is the 1-dimensional affine subspace of $\mathbb{F}_2^2$ satisfying $x' = x$.

We can't generally expect to compute precise subspace abstractions in polynomial time, even for circuits which implement affine transitions, as a polynomial-time solution to the problem of *strong simulation* — computing $\langle \vec{x}' | U |x\rangle$ given a circuit U — would imply **BQP** = **P** [15]. While the problem of determining a precise abstraction of a circuit as an affine transformation is slightly weaker in that it only tells us when $\langle \vec{x}' | U |x\rangle \neq 0$, it can be noted that this suffices to efficiently simulate several *deterministic* quantum algorithms which exhibit exponential query complexity separation between quantum and classical algorithms [8, 45].

The following proposition establishes a basis for merging phase gates over affine transition relations. In particular, if $x'_j = x_i$ in every non-zero transition of a control-flow path, then a phase gate on qubit i in the prefix may be merged with a phase gate on qubit j in the suffix.

Proposition 9. *Let $\mathcal{S}[\vec{X}';\ \vec{X}]$ be a sound abstraction of a path π and suppose that for every $(\vec{x}', \vec{x}) \in \mathcal{S}$, $x'_j = x_i$. Then $\llbracket \mathrm{R}_Z(\theta)_{q_i} \fatsemi \pi \fatsemi \mathrm{R}_Z(\phi)_{q_j} \rrbracket = \llbracket \mathrm{R}_Z(\theta+\phi)_{q_i} \fatsemi \pi \fatsemi \mathbf{skip} \rrbracket = \llbracket \mathbf{skip} \fatsemi \pi \fatsemi \mathrm{R}_Z(\theta+\phi)_{q_j} \rrbracket$.*

Proof. By computation, noting that $\vec{x}' \in \mathrm{supp}(\llbracket \pi \rrbracket |\vec{x}\rangle)$ only if $(\vec{x}', \vec{x}) \in \mathcal{S}$ and hence $x'_j = x_i$:

$$\begin{aligned}
\llbracket \mathrm{R}_Z(\theta)_{q_i} \fatsemi \pi \fatsemi \mathrm{R}_Z(\phi)_{q_j} \rrbracket |\vec{x}\rangle &= e^{i\theta x_i} \left(\mathrm{R}_Z(\phi)_{q_j} \llbracket \pi \rrbracket |\vec{x}\rangle \right) \\
&= e^{i\theta x_i} \left(\mathrm{R}_Z(\phi)_{q_j} \sum\nolimits_{\vec{x}' \in \mathrm{supp}(\llbracket \pi \rrbracket |\vec{x}\rangle)} \alpha_{\vec{x}'} |\vec{x}'\rangle \right) \\
&= e^{i\theta x_i} \left(\sum\nolimits_{\vec{x}' \in \mathrm{supp}(\llbracket \pi \rrbracket |\vec{x}\rangle)} e^{i\phi x'_j} \alpha_{\vec{x}'} |\vec{x}'\rangle \right) \\
&= e^{i(\theta+\phi)x_i} \llbracket \pi \rrbracket |\vec{x}\rangle
\end{aligned}$$

The claim then follows by linearity. □

Note that Proposition 9 gives a method of merging phases applied *along* a path. In particular, if we view x_i and x'_j as linear functionals $f_i(\vec{x}', \vec{x}) = x_i$, $f_j(\vec{x}', \vec{x}) = x'_j$, respectively, then we may represent the *condition* $f \in (\mathbb{F}_2^{2n})^*$ of each phase gate via a canonical representative (if it exists) on the subspace $\mathcal{S}[\vec{X}';\ \vec{X}]$. If two conditions $f_i + f_j \in \ker \mathcal{S}[\vec{X}';\ \vec{X}]$ where $\ker \mathcal{S}$ is the subspace of $(\mathbb{F}_2^{2n})^*$ which annihilates $\mathcal{S}$, then by definition $f_i(\vec{x}', \vec{x}) = f_j(\vec{x}', \vec{x})$ for all $(\vec{x}', \vec{x}) \in \mathcal{S}$.

$$\mathcal{A}[\![\mathrm{X}]\!] = \langle x'_q \oplus 1 \oplus x_q \rangle \qquad \mathcal{A}[\![q := |0\rangle]\!] = \langle x'_q \rangle$$
$$\mathcal{A}[\![\mathrm{R}_Z(\theta)_q]\!] = \langle x'_q \oplus x_q \rangle \qquad \mathcal{A}[\![\textbf{meas } q]\!] = \langle x'_q \oplus x_q \rangle$$
$$\mathcal{A}[\![\mathrm{H}_q]\!] = \top_q \qquad \mathcal{A}[\![\textbf{call } p(\vec{q})]\!] = \mathcal{A}[\![p(\vec{q})]\!]$$
$$\mathcal{A}[\![\mathrm{CNOT}_{q_1 q_2}]\!] = \langle x'_{q_1} \oplus x_{q_1}, x'_{q_2} \oplus x_{q_1} \oplus x_{q_2} \rangle \qquad \mathcal{A}[\![T_1;\ T_2]\!] = \mathcal{A}[\![T_1]\!] \fatsemi \mathcal{A}[\![T_2]\!]$$
$$\mathcal{A}[\![P^b_q]\!] = \langle x'_q \oplus x_q, x_q \oplus b \rangle \qquad \mathcal{A}[\![\textbf{if } \star \textbf{ then } T_1 \textbf{ else } T_2]\!] = \mathcal{A}[\![T_1]\!] \sqcup \mathcal{A}[\![T_2]\!]$$
$$\mathcal{A}[\![U\vec{q}]\!] = \mathcal{A}[\![U]\!][X'_{\vec{q}};\ X_{\vec{q}}] \qquad \mathcal{A}[\![\textbf{while } \star \textbf{ do } T]\!] = \mathcal{A}[\![T]\!]^\star$$

Fig. 7. Affine relation analysis for quantum WHILE programs

4.3 Extending to Quantum Programs

The developments of the previous section allow the phase folding algorithm to be easily extended to non-deterministic quantum WHILE programs. Given a set of paths Π, a two-vocabulary subspace $\mathcal{S} \subseteq \mathbb{F}_2^{2n}$ is a sound abstraction of Π if and only if $\mathcal{S}$ is a sound abstraction of every $\pi \in \Pi$. Correctness of the phase folding condition, i.e. Proposition 9, carries over to sets of paths in this way, as phase gates may be merged if and only if they can be merged along every path.

Given a concrete domain of subspaces of $\mathbb{F}_2^{2n}$, we may compute abstractions of sets of paths from abstractions of individual paths in the standard way by taking the *join*, corresponding to the *affine hull* of subspaces. In particular, if $\mathcal{S}_1$ and $\mathcal{S}_2$ are sound abstractions of π_1 and π_2, it is easy to see that the smallest subspace which soundly abstracts both is the affine hull of $\mathcal{S}_1$ and $\mathcal{S}_2$, denoted $\mathcal{S}_1 \sqcup \mathcal{S}_2$ where

$$\mathcal{S}_1 \sqcup \mathcal{S}_2 \coloneqq \mathrm{span}(\{\vec{x} \mid \vec{x} \in \mathcal{S}_1 \cup \mathcal{S}_2\})$$

Moreover, as noted by [31], since the subspaces of $\mathbb{F}_2^{2n}$ have finite and bounded dimension, there can be no infinite strictly ascending chain $\mathcal{S}_1 \subsetneq \mathcal{S}_2 \subsetneq \cdots$ and so the join of infinitely many paths stabilizes in finitely many ($O(n)$) steps. As is customary [32] we define the *Kleene closure* $\mathcal{S}^\star$ of a two-vocabulary subspace $\mathcal{S}[\vec{X}';\ \vec{X}]$ to be the limit of the sequence $\{\mathcal{S}_i\}_{i=0}^{\infty}$ where

$$\mathcal{S}_0 = \bot = \{0\} \qquad \mathcal{S}_{i+1} = \mathcal{S}_i \sqcup (\mathcal{S}_i \fatsemi \mathcal{S})$$

which satisfies $\mathcal{S}_i \subseteq \mathcal{S}_{i+1}$ for all $i \geq 0$ and hence stabilizes in at most $2n$ steps.

We now have all the pieces necessary to define an affine relation analysis for quantum WHILE programs. Let

$$(\mathcal{S}(\mathbb{F}_2^{2n}), \subseteq, \sqcup, \sqcap, \bot, \top)$$

be some domain of affine transition relations where $\mathcal{S}(\mathbb{F}_2^{2n})$ is the set of affine subspaces of $\mathbb{F}_2^{2n}$ with a special element $\bot$ for the empty subset, $\sqcup$ is the union of affine subspaces, $\sqcap$ is subspace intersection, and $\top = \mathbb{F}_2^{2n}$. Additionally, let $\mathcal{S}(\mathbb{F}_2^{2n})$ be equipped with an additional composition operator $\fatsemi$. Figure 7 gives an abstract semantics $\mathcal{A}[\![T]\!] \in \mathcal{S}(\mathbb{F}_2^{2n})$ for computing a sound approximation of a quantum WHILE program over typical gates as an affine transition function. Transition relations for gates and projections are given as a function of the qubits they act on — we naturally extend the transition relation of a gate on qubits $\vec{q}$ to a transition relation on Q by setting the pre- and post-state for any qubit in $Q - \vec{q}$ to be equal.

The analysis of Figure 7 is defined without respect to a particular implementation of the subspace domain. In principle any *relational* domain for affine subspaces suffices, the key element of relational in this case being the ability to compose relations and hence abstract affine *transfer* functions rather than sets of states. The **KS** domain of [18] is one such concrete implementation which suffices for

the purpose of quantum ARA. In the next section we develop a concrete domain for performing phase folding with an affine relation analysis which is based on the **KS** domain.

Theorem 10. *Let T be a quantum WHILE program. Then $\mathcal{A}[\![T]\!]$ is a sound abstraction of $\mathcal{T}[\![T]\!]$.*

Proof. It can be readily observed that the abstractions of basic gates are all sound (and precise), and likewise that the rules for unitary gate application, composition, non-deterministic choice and iteration are sound. It remains to show that the transition relations for reset and measurement are sound. For measurement, we note that $\mathcal{T}[\![\textbf{meas } q]\!] = \{\textbf{assume } P_q^0\} \cup \{\textbf{assume } P_q^1\}$, and hence a sound abstraction of **meas** q is $\mathcal{A}[\![P^0]\!] \sqcup \mathcal{A}[\![P^1]\!] = \langle x' \oplus x, x\rangle \sqcup \langle x' \oplus x, x \oplus 1\rangle = \langle x' \oplus x\rangle$. Likewise, for reset we have $\mathcal{T}[\![q := |0\rangle]\!] = \{\textbf{assume } P_q^0\} \cup \{\textbf{assume } P_q^1 \fatsemi Xq\}$, which is soundly abstracted by the join $\mathcal{A}[\![P^0]\!] \sqcup \mathcal{A}[\![XP^1]\!] = \langle x' \oplus x, x\rangle \sqcup \langle x', x \oplus 1\rangle = \langle x'\rangle$. □

4.4 Forward ARA with Summarization

We now give a concrete algorithm for phase folding with respect to affine transition relations over quantum WHILE programs. We use the **KS** domain of [18, 33] for the affine relational analysis, tailored to the pecularities of the phase folding context. Notably,

(1) programs are *large*: most quantum programs include large sections of straightline code involving thousands, if not millions, of gates. To handle practical examples of quantum circuits our analysis should scale to $m = 10^3$–10^6 gates.
(2) Projection is *expensive*: for a circuit involving m gates, there are $O(m)$ phase terms which need to be normalized when projecting out a variable.
(3) Variables are *inexpensive*: as our variables are binary, we can implement fast linear algebra over many variables by storing (row) vectors as bitvectors.

To this end, our analysis is designed to avoid projection as much as possible, and to avoid whenever possible applying projection to phase conditions. This amounts to performing a *forward analysis* which maintains the current state as an affine subspace over the pre-state and intermediate variables, together with *summarization* of loops and branches using **KS** domain operations.

We first recall the definition of the **KS** domain from [18]. An element of the **KS** domain in vocabularies $\vec{X}', \vec{X}, \vec{Y}$, denoted $\mathbf{KS}[\vec{X}';\ \vec{X};\ \vec{Y}]$ where $|\vec{X}| = n$ and $|\vec{Y}| = k$, is a matrix $A \in \mathbb{F}_2^{m\times 2n+k+1}$ in reduced echelon form with all-zero rows dropped. An element of the domain represents an affine 3-vocabulary subspace $\mathcal{S}[\vec{X}';\ \vec{X};\ \vec{Y}]$ of $\mathbb{F}_2^{2n+k}$ as the (affine) kernel of A, $\ker A$, and its concretization is equal to this subspace. The rows of A are viewed as affine constraints $A_i(\vec{x}', \vec{x}, \vec{y}) + c_i = 0$ where the first n columns correspond to variables $\vec{X}'$, followed by the variables $\vec{X}$ and then the variables $\vec{Y}$. A special 1-row element, $\bot = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 1 \end{array}\right]$ is reserved to represent the empty subspace, and whenever A contains such a row which is necessarily the case when $\ker A = \emptyset$, A is automatically reduced to $\bot$. The top element $\top$ is the empty matrix corresponding uniquely to $\ker \top = \mathbb{F}_2^{2n+k}$.

Projection of $\vec{X}'$ for an element A of $\mathbf{KS}[\vec{X}';\ \vec{X};\ \vec{Y}]$ is implemented by removing any row which is non-zero in one of the columns of $\vec{X}'$. Other variable sets may be projected out by re-ordering the columns to place the variables to be projected in the left-most columns, writing in row-echelon with the new variable order, and then projecting out those columns. The remaining domain operations can be defined via projection as below, where $A = \left[\begin{array}{ccc|c} A_{\text{Post}} & A_{\text{Post}} & A_{\text{Aux}} & \vec{c} \end{array}\right]$, $B = \left[\begin{array}{ccc|c} B_{\text{Post}} & B_{\text{Post}} & B_{\text{Aux}} & \vec{d} \end{array}\right]$ are elements of the **KS** domain and $\exists.$ denotes the projection of the left-most block:

$$A \sqcap B = \begin{bmatrix} A \\ B \end{bmatrix} \qquad A \sqcup B = \exists. \left[\begin{array}{c|c} A & A \\ B & 0 \end{array}\right] \qquad A \fatsemi B = \exists. \left[\begin{array}{cc|ccc|c} A_{\text{Post}} & 0 & A_{\text{Pre}} & A_{\text{Aux}} & 0 & \vec{c} \\ B_{\text{Pre}} & B_{\text{Post}} & 0 & 0 & B_{\text{Aux}} & \vec{d} \end{array}\right]$$

$$\mathbf{PF}_{\text{Aff}}[\![\text{H}_q]\!](t,A) = (t, A[x'_q \leftarrow y_{k+1} \oplus 1]) \qquad \mathbf{PF}_{\text{Aff}}[\![\text{X}_q]\!](t,A) = (t, A[x'_q \leftarrow x_q \oplus 1])$$
$$\mathbf{PF}_{\text{Aff}}[\![\text{CNOT}_{q_1q_2}]\!](t,A) = (t, A[x'_{q_2} \leftarrow x_{q_1} \oplus x_{q_2}]) \qquad \mathbf{PF}_{\text{Aff}}[\![q := |0\rangle]\!](t,A) = (t, A[x'_q \leftarrow 0])$$
$$\mathbf{PF}_{\text{Aff}}[\![\text{R}_Z(\theta)^\ell_q]\!](t,A) = (t \uplus ((A^{\text{Pre+Post}})_i, \{\ell\}), A) \qquad \mathbf{PF}_{\text{Aff}}[\![\textbf{meas } q]\!](t,A) = (t,A)$$
$$t_1 \uplus t_2 = \{(\bot, L) \mid (\bot, L) \in t_1 \cup t_2\} \cup \{(f, L_1 \cup L_2) \mid (f, L_1) \in t_1 \wedge (f, L_2) \in t_2\}$$

Fig. 8. Forward analysis of basic blocks.

We define one additional operation on the **KS** domain,

$$\text{reduce} : \mathbf{KS}[\vec{X}';\ \vec{X};\ \vec{Y}] \times (\mathbb{F}_2^{2n+k})^* \to (\mathbb{F}_2^{2n+k})^*$$

where $(\mathbb{F}_2^{2n+k})^*$ denotes the dual space of $\mathbb{F}_2^{2n+k}$. The operation reduce canonicalizes a linear functional $f \in (\mathbb{F}_2^{2n+k})^*$ with respect to an element A of $\mathbf{KS}[\vec{X}';\ \vec{X};\ \vec{Y}]$ by row reducing the matrix $\left[\begin{smallmatrix} A \\ f \end{smallmatrix}\right] \to \left[\begin{smallmatrix} A \\ f' \end{smallmatrix}\right]$ and returning f'. Note that $f_1(\vec{x}', \vec{x}, \vec{y}) = f_2(\vec{x}', \vec{x}, \vec{y})$ for all $(\vec{x}', \vec{x}, \vec{y}) \in \ker A$ if and only if $\text{reduce}(A, f_1) = \text{reduce}(A, f_2)$, since the reduction of either gives a unique representation of the kernel of $\left[\begin{smallmatrix} A \\ f_1 \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} A \\ f_2 \end{smallmatrix}\right]$, respectively. Note that in contrast to [18], we write **KS** elements with the post-state first so that ordinary Gaussian elimination will eliminate those first, as our goal is to canonicalize over the pre- and intermediate- states. In practice, the pre- and intermediate-states are separable in our analysis, so their order does not matter.

We can now describe the phase folding analysis. We denote the set of program locations by Loc. A *condition* $\text{Cond} := (\mathbb{F}_2^{2n+k})^* \cup \{\bot\}$ is either $\bot$ or $f \in (\mathbb{F}_2^{2n+k})^*$, and a *phase term* is a pair $f(, L)$ of a condition f and a subset of program locations L. We define

$$\mathbf{PF}_{\text{Aff}} = \mathcal{P}(\text{Cond} \times \mathcal{P}(\text{Loc})) \times \mathbf{KS}[\vec{X}';\ \vec{X};\ \vec{Y}].$$

The intuition behind an element of $\mathbf{PF}_{\text{Aff}}$ is that it represents as program T as an affine subspace on the pre-, post-, and intermediate-states, along with a partition of the R_Z gates of T into disjoint sets, each of which has a representative expressing the condition as a linear function of the pre-, post-, and intermediate-states. If no such function exists, the set has representative $\bot$.

The algorithm proceeds in an iterative fashion for basic blocks. The transfer functions $\mathbf{PF}_{\text{Aff}}[\![T]\!] : \mathbf{PF}_{\text{Aff}} \to \mathbf{PF}_{\text{Aff}}$ are given in Figure 8. The forward analysis maintains an element $(t, A) \in \mathbf{PF}_{\text{Aff}}$ where $A = \left[\begin{smallmatrix} I & A_{\text{Pre}} & A_{\text{Aux}} \mid \vec{c} \end{smallmatrix}\right]$ so that $\vec{x}' = A_{\text{Pre}}\vec{x} + A_{\text{Aux}}\vec{y} + \vec{c}$. When a phase gate is applied to qubit i at location ℓ, ℓ is added to t with condition $f = (A^{\text{Pre+Post}})_i$ given by the ith row of A with the (irrelevant) affine factor dropped. If another phase term has the same (non-$\bot$) representative function f, the phase terms are merged by taking the union of their locations (denoted $\uplus$ in Figure 8). Gates with transitions for which some x'_i has no representative in terms of X and Y, notably the H gate, add a new intermediate variable $Y \cup \{y_{k+1}\}$ and are modeled with the relation $x'_i = y_{k+1}$ to maintain A in the above form. The initial value is defined as $(\emptyset, I)$.

Computing summaries. To deal with choice, iteration, and procedure calls, the algorithm first analyzes the body or bodies and then produces a *summary*. Figure 9 gives our summarization rules, where $(t_1, A_1) \sqcup (t_2, A_2) := (t_1 \cup t_2, A_1 \sqcup A_2)$ and $(t, A)^\star := (t, A^\star)$. We distinguish summaries from iterative values of $\mathbf{PF}_{\text{Aff}}$ as the summary generally won't be of a form where $\vec{x}'$ has an explicit representation in terms of $\vec{x}$ and $\vec{y}$ — we say such relations are *solvable*. For instance, at the end of a loop $\vec{x}'$ may depend on a temporary value initialized in the loop body, which must then be projected away to compute the Kleene closure. Moreover, phase gates applied inside a loop, conditional, or procedure can't cross the control-flow boundaries even if they apply on the same condition as some phase gate outside the loop. The exception is phase gates conditional on the zero-everywhere

$$\mathbf{PF}_{\mathrm{Sum}}[\![\textbf{if } \star \textbf{ then } T_1 \textbf{ else } T_2]\!] = \mathbf{PF}_{\mathrm{Aff}}[\![T_1]\!](\emptyset, I) \sqcup \mathbf{PF}_{\mathrm{Sum}}[\![T_2]\!](\emptyset, I)$$
$$\mathbf{PF}_{\mathrm{Sum}}[\![\textbf{while } \star \textbf{ do } T]\!] = \mathbf{PF}_{\mathrm{Aff}}[\![T]\!](\emptyset, I)^\star$$
$$\mathbf{PF}_{\mathrm{Sum}}[\![\textbf{call } p(\vec{q})]\!] = (\{(\bot, L) \mid (-, L) \in t\}, A[X'_{\vec{q}}, X_{\vec{q}}]) \qquad \text{where}(t, A) = \mathbf{PF}_{\mathrm{Aff}}[\![T]\!](\emptyset, I)$$

Fig. 9. Summary computations for choice, iteration, and procedure calls.

$$\mathbf{PF}_{\mathrm{ff}}(t, A)(t', A') = (t \uplus t'_{\mathrm{null}}, A \fatsemi_{\mathrm{ff}} A'),\ t'_{\mathrm{null}} = \left\{ (r, L) \mid (f, L) \in t' \wedge r = \begin{cases} 0 & \text{if reduce}(A, f) = 0 \\ \bot & \text{otherwise} \end{cases} \right\}$$

Fig. 10. Summary application.

function, $\vec{0}^T$, which can safely be eliminated — to identify these cases, the summary preserves any partitions whose condition is *nullable* when composed with a pre-condition, and maps the representative of every other partition to $\bot$.

Applying summaries. To apply summaries, we must compose a solvable relation A with some potentially non-solvable relation A' into a once again solvable relation. To do so, we add an extra vocabulary of intermediate variables $\vec{Y}'$ and assert the equality between the post-state $\vec{X}'$ and $\vec{Y}'$. Explicitly, we define the summary composition operator $\fatsemi_{\mathrm{ff}}$ as

$$A \fatsemi_{\mathrm{ff}} A' = A \fatsemi A' \sqcap \langle \vec{X}' = \vec{Y}' \rangle = \exists. \left[\begin{array}{c|ccccc|c} A_{\mathrm{Post}} & 0 & A_{\mathrm{Pre}} & A_{\mathrm{Aux}} & 0 & \vec{c} \\ A'_{\mathrm{Pre}} & A'_{\mathrm{Post}} & 0 & 0 & 0 & \vec{d} \\ 0 & I & 0 & 0 & I & \vec{0} \end{array}\right]$$

In many practical contexts such as in Grover's algorithm, an ancilla is often initialized in the $|0\rangle$ state prior to entering a loop, with the invariant that the ancilla is returned to the initial state after each iteration. Such situations can be used to perform optimizations *inside* the loop body, particularly if the assumption that the ancilla is in the $|0\rangle$ leads to a phase condition becoming null (i.e. $f = 0$). To make use of these optimizations, when composing a domain with a summary for a block we check whether the pre-state defined by the domain element implies any of the phase gates in the summarized block are nullable. To do so, we reduce the phase terms inside the summarized block with the pre-state and record the phases whose conditions have become null, as in Figure 10.

Theorem 11. *Let* $\mathbf{PF}_{\mathrm{Aff}}[\![T]\!] = (t, A)$. *Then for every* $(f, L) \in t$, *the phase gates at locations* $\ell \in L$ *in* T *can be safely merged into a single gate, or eliminated if* $f = 0$. *Moreover,* $\mathbf{PF}_{\mathrm{Aff}}[\![T]\!]$ *can be computed in time* $O(n^3 m + nm^2 \log m)$ *where* n *is the number of qubits and* m *is the number of program locations.*

Proof. Correctness follows from Theorem 10 and Proposition 9. For the complexity we assume that bit vectors are stored as packed integers, hence operations on bit vectors take time $O(1)$. At each step in the analysis, t has at most m terms, and A is an $n \times 2n + k + 1$ matrix represented as a list of n bit vectors. Row operations corresponding to gate applications hence take $O(1)$ time, adding to t takes time $O(\log m)$, while Gaussian elimination for computing joins takes $O(n^2)$ time. Loop summaries stabilize in at most n iterations, since the initial dimension of the loop transition subspace is n, giving a total cost of $O(n^3)$ to compute a loop summary. Finally, applying a summary involves a Gaussian elimination at cost $O(n^2)$ as well as reduction of all terms of t, each at a cost of $O(n)$ for a total cost of $O(nm \log m)$ to reduce and merge all terms of t. The resulting complexity is hence bounded by $O(n^3 m + nm^2 \log m)$. □

5 Non-linear Phase Folding

Re-casting the phase folding optimization as a relational analysis not only allows the original *affine* analysis to be generalized to quantum programs, but also allows the optimization to be easily generalized to *non-linear* analyses. Such analyses can be used to capture instances of phase folding where the non-linear Toffoli gate TOF gate is used like in the motivating example, as well as situations where affine transition relations give rise to complicated non-linear loop invariants. Note that by non-linear here we mean that the Toffoli gate, while linear as a unitary operator over $\mathbb{C}^{2^3}$, computes a non-linear permutation of $\mathbb{F}_2^3$ expressed as the relation $\textsc{tof} : |x, y, z\rangle \mapsto |x, y, z \oplus xy\rangle$. In this section we present one such analysis based on abstracting classical transitions via *polynomial ideals* which suffices to achieve both optimizations. While the connection between polynomial ideals and loop invariants has been well-established [14, 44, 47], the simplistic nature of the underlying finite field $\mathbb{F}_2$ allows us to do away with most of the complications involved in similar classical program analyses. Moreover, our techniques are illustrative of the flexibility of our algorithm in making use of classical techniques for relational analysis.

5.1 Polynomial Transition Ideals

We first recall some basic details of algebraic geometry [12]. An *affine variety* over $\mathbb{F}_2^n$ is the set of simultaneous solutions of a set $S \subseteq \mathbb{F}_2[X_1 \ldots, X_n]$ of polynomial equations:

$$V = \mathbb{V}(S) = \{\vec{x} \in \mathbb{F}_2^n \mid f(\vec{x}) = 0 \ \forall f \in S\}.$$

It is a basic fact that the polynomials which vanish on the variety V form an ideal I in $\mathbb{F}_2[\vec{X}]$,

$$I = \mathbb{I}(V) = \{f \in \mathbb{F}_2[\vec{X}] \mid f(\vec{x}) = 0 \ \forall \vec{x} \in V\}.$$

Given an algebraic variety V over $\vec{X}$, the polynomial ideal $\mathbb{I}(V)$ can be viewed as the set of *polynomial relations on* $\vec{X}$ which are satisfied by V, in the same way that an affine *subspace* can be viewed as a set of affine (degree 1) *relations* satisfied by elements of the subspace, as in particular $\mathbb{V}(\mathbb{I}(V)) = V$. We say that an ideal I has a *basis* if $I = \langle S\rangle$ for some $S \subseteq \mathbb{F}_2[\vec{X}]$. By Hilbert's basis theorem, an ideal in $\mathbb{F}_2[\vec{X}]$ necessarily has a basis. Note that $\mathbb{V}(S) = \mathbb{V}(\langle S\rangle)$, and hence for a variety V defined by a set of polynomials S it suffices to represent V as $\mathbb{V}(\langle S\rangle)$. The union and intersection of varities corresponds to the product and sum of their ideals, respectively:

$$\mathbb{V}(I) \cup \mathbb{V}(J) = \mathbb{V}(I \cdot J) \qquad \mathbb{V}(I) \cap \mathbb{V}(J) = \mathbb{V}(I + J).$$

Moreover, given bases for I and J we can construct a basis for $I \cdot J$ and $I + J$, as

$$\langle S\rangle \cdot \langle T\rangle = \langle\{fg \mid f \in S, g \in T\}\rangle \qquad \langle S\rangle + \langle T\rangle = \langle S \cup T\rangle.$$

We will be interested in a particular type of basis called a Gröbner basis, defined with respect to a monomial order. Recall that a *monomial order* $\succcurlyeq$ is a total well-order on monomials m, n such that for any monomials m, n, w, $m \succcurlyeq n \iff mw \succcurlyeq nw$. A Gröbner basis for a polynomial ideal I over the monomial ordering $\succcurlyeq$ is a particular type of basis for I over which reduction mod I yields unique normal forms. For a particular monomial order, the *reduced* Gröbner basis of an ideal, obtained by mutually reducing each element of the Gröbner basis with respect to one another, is unique. Methods for computing (reduced) Gröbner bases exist [11, 19] and have been the subject of much study in symbolic computation — though they require *exponential* time unlike the Gaussian elimination used in ARA.

Given a polynomial ideal I in disjoint variable sets $\vec{X}, \vec{Y}$, the *elimination ideal* of $\vec{X}$ is $I_{\vec{Y}} := I \cap \mathbb{F}_2[\vec{Y}]$ which can be viewed as the polynomial relations on $\vec{Y}$ which are implied by I. Elimination ideals, which correspond to projection from the perspective of affine varieties, can be conveniently computed via Gröbner basis methods. An elimination order for $\vec{X}$ is a monomial order such that

any monomial involving a variable in $\vec{X}$ occurs earlier in the order than a monomial which does not contain a variable in $\vec{X}$. A standard result states that if G is a Gröbner basis for some ideal $I \subseteq \mathbb{F}_2[\vec{X}, \vec{Y}]$ in a monomial order eliminating $\vec{X}$, then $G \cap \mathbb{F}_2[\vec{Y}]$ is a Gröbner basis for $I \cap \mathbb{F}_2[\vec{Y}]$.

As a final point we consider whether an ideal I contains *all* polynomial relations implied by the affine variety $\mathbb{V}(I)$. In the case of algebraically closed fields this is the case, as $\mathbb{I}(\mathbb{V}(I)) = I$ due to Hilbert's Nullstellensatz. While $\mathbb{F}_2$ is not algebraically closed and hence the Nullstellensatz fails, when considering *multilinear* ideals — ideals in $\mathbb{F}_2[\vec{X}]/\langle X_i^2 - X_i \mid X_i \in \vec{X}\rangle$ — we can recover equality from Hilbert's *strong* Nullstellensatz. Given an ideal I in some polynomial ring, the *radical* of I, denoted $\sqrt{I}$ is defined as the ideal

$$\{f \mid f^m \in I \text{ for some integer } m \geq 1\}$$

Hilbert's *strong* Nullstellensatz asserts that $\mathbb{I}(\mathbb{V}(I)) = \sqrt{I}$. As shown in [24], over $\mathbb{F}_2$ an ideal's radical is obtained by adjoining the field equations $X_i^2 - X_i$, or equivalently reduce the ideal to *multilinear* polynomials.

Proposition 12 ([24]). *Let I be an ideal over $\mathbb{F}_2[\vec{X}]$. Then*

$$\mathbb{I}(\mathbb{V}(I)) = \sqrt{I} = I + \langle X_i^2 - X_i \mid X_i \in \vec{X}\rangle.$$

Proposition 12 implies in particular that our straightforward methods are in fact *precise, despite working over a non-algebraically closed field.*

5.2 The Pol Domain

Given a unitary U such that $U|\vec{x}\rangle = |f_1(\vec{x}), f_2(\vec{x}), \ldots, f_n(\vec{x})\rangle$ where f_i are polynomial equations, the set of non-zero transitions $\langle \vec{x}'| U |\vec{x}\rangle \neq 0$ of U forms an affine variety over $\mathbb{F}_2^{2n}$ — in particular, $\vec{x'} \in \text{supp}(U|\vec{x}\rangle)$ if and only if $(\vec{x}', \vec{x}) \in \mathbb{V}(\{X_1' \oplus f_1(\vec{X}), \ldots, X_n' \oplus f_n(\vec{X})\})$. Given an affine variety V in two vocabularies $\vec{X}', \vec{X}$ which contains all non-zero transitions of a quantum WHILE program, we may hence represent V precisely as the set of polynomial relations $I = \mathbb{I}(V)$. We call this the $\mathbf{Pol}[\vec{X}';\ \vec{X}]$ domain, whose elements are (Boolean) polynomial ideals I over two (or more) vocabularies and whose concretization is given by $\mathbb{V}(I)$. We implicitly adjoin the field equations $I_0 = \langle X_i^2 - X_i, X_i'^2 - X_i' \mid X_i \in \vec{X}\rangle$ to elements of **Pol** "under the hood" as is standard in dealing with ideals over $\mathbb{F}_2$ [10].

Elements I, J of **Pol** are stored as reduced Göbner bases, which allows checking equality of transition ideals. The domain **Pol** is naturally equipped with an order corresponding to the (reverse) subset relation on ideals $I \sqsubseteq J \iff I \supseteq J$, as well as greatest and least elements $\top = \emptyset$ and $\bot = \langle 1\rangle$, respectively, where $\top$ is the *least* precise element. The reverse order is chosen so that joins correspond to the union of varieties. In that regard we define $I \sqcup J = I \cdot J$ and $I \sqcap J = I + J$, where the concrete representation is again as a reduced Gröbner basis. Projection $\exists \vec{X}.I[\vec{X};\ \vec{Y}]$ corresponds to the elimination ideal $I_{\vec{Y}}$, which as discussed in the previous section can be implemented by computing a Gröbner basis over an elimination order and projecting out $\vec{X}$. We can then define sequential composition as in affine subspaces as

$$I[\vec{X}';\ \vec{X}] \fatsemi J[\vec{X}';\ \vec{X}] = \exists \vec{X}''.I[\vec{X}'';\ \vec{X}] \sqcap J[\vec{X}';\ \vec{X}''].$$

The identity transition relation $1 = \langle \vec{X}' \oplus \vec{X}\rangle$ once again serves as the neutral element with respect to composition.

Kleene iteration is surprisingly easy to define in our case, as we are working over the finite field $\mathbb{F}_2$ with polynomials which are in fact multilinear. Specifically, it can be observed that $\mathbb{F}_2[\vec{X}', \vec{X}]/I_0$ is finite, and so any ascending chain of ideals over this ring necessarily stabilizes. We can hence

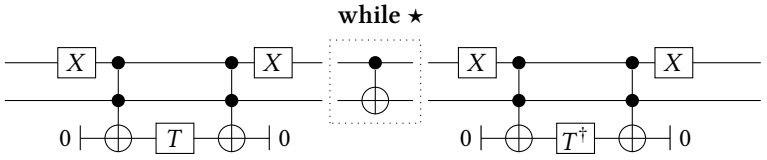


Fig. 11. A circuit with eliminable T gates. The loop satisfies the invariant $(1 \oplus x_0')x_1' = (1 \oplus x_0)x_1$.

define the Kleene iteration $I^\star$ of an ideal similarity to the affine subspace domain as the limit of the sequence $\{I_i\}_{i=0}^{\infty}$ where

$$I_0 = \bot \qquad\qquad I_{i+1} = I_i \sqcup (I_i \fatsemi I)$$

which again satisfies $I_i \sqsubseteq I_{i+1}$ for all $i \geq 0$. This avoids the need for solving recurrence relations as in most work on polynomial invariant generation [14, 44], and is a significantly simplifying assumption which leads to easy implementation of non-linear reasoning in quantum programs.

Remark 13. We use the ideal product for joins rather than the generally more precise ideal *intersection* $I \cap J$, as the product is generally faster to compute [12], and by Proposition 12 and the fact that $\sqrt{I \cdot J} = \sqrt{I \cap J}$ asserts that the product is equal to the intersection in our case. We ran experiments with both and while we did not find a significant difference in computation time, they consistently produced the same ideal every time as expected.

5.3 A Non-linear Phase Folding Algorithm

The **Pol** domain now suffices for a drop-in replacement of the affine **KS** domain in the phase folding algorithm of Section 4.4. We extend conditions of phases from linear functionals $f(\vec{x})$ to *polynomial* equations $f(\vec{x})$, of which the linear functionals form a subset. Reduction of a phase condition modulo a transition ideal I represented as a reduced Gröbner basis G amounts to multivariate polynomial division by G which was previously noted to produce a unique canonical form $f' \in \mathbb{F}_2[\vec{X}';\ \vec{X}]/I$. We denote the resulting analysis $\mathbf{PF}_{\mathrm{Pol}}[\![\cdot]\!]$.

Theorem 14. *Let* $\mathbf{PF}_{\mathrm{Pol}}[\![T]\!] = (t, I)$. *Then for every* $(f, L) \in t$, *the phase gates at locations* $\ell \in L$ *in* T *can be safely merged into a single gate, or eliminated if* $f = 0$.

Example 15. Figure 11 gives an example of a looping quantum program which can be optimized by using a non-linear loop invariant. The loop satisfies the invariant $(1 \oplus x_0')x_1' = (1 \oplus x_0)x_1$, as the second qubit is not modified if $x_0 = 0$. Noting that the T gates apply phases of $\omega^{(1\oplus x_0)x_1}$ and $(\overline{\omega})^{(1\oplus x_0')x_1'}$, respectively, the total phase contributed is $(\omega\overline{\omega})^{(1\oplus x_0)x_1} = 1$ and both can hence be eliminated.

Our algorithm proceeds to compute the loop invariant by first constructing the transition ideal $\langle x_0' \oplus x_0, x_1' \oplus x_0 \oplus x_1\rangle$ of the loop body, corresponding to the CNOT : $|x_0, x_1\rangle \mapsto |x_0, x_1 \oplus x_0\rangle$ gate, and then taking the Kleene closure

$$\langle x_0' \oplus x_0, x_1' \oplus x_1 \oplus x_0\rangle^\star = \langle x_0' \oplus x_0, x_1' \oplus x_1\rangle \sqcup \langle x_0' \oplus x_0, x_1' \oplus x_1 \oplus x_0\rangle.$$

Computing a Gröbner basis of the right hand side gives $\langle x_0' \oplus x_0, x_1' \oplus x_1 \oplus x_0x_1' \oplus x_0x_1\rangle$, which notably reduces $(1 \oplus x_0')x_1'$ to $(1 \oplus x_0)x_1$. After eliminating both T gates, the remaining gates can be canceled via basic gate cancellations.

6 Increasing the Precision of Abstract Transformers

It was previously noted that composition of transition relations (affine or otherwise) does *not* produce the most precise relation for a control-flow path, due to the effects of *interference* on

classical states *in superposition*. In this section, we show that precise transition relations of entire circuits can be generated automatically by symbolically identifying and reducing interfering paths using the *sum-over-paths* technique. Such techniques are useful for generating abstract transformers for gates which are defined as circuits over some small set of basic gates.

6.1 Symbolic Path Integrals

A *path integral* or *sum* is a representation of a linear operator $\Psi : \mathbb{C}^{2^n} \to \mathbb{C}^{2^m}$ having the form

$$\Psi : |\vec{x}\rangle \mapsto \sum_{\vec{y} \in \mathbb{F}_2^k} \Phi(\vec{x}, \vec{y}) \, |f(\vec{x}, \vec{y})\rangle \tag{1}$$

where $\Phi : \mathbb{F}_2^{n+k} \to \mathbb{C}$ and $f : \mathbb{F}_2^{n+k} \to \mathbb{F}_2^n$ are the *amplitude* and *transition* functions, respectively. Intuitively, a path sum represents a linear operator (e.g. a quantum circuit) as a set of *classical* transitions or *paths* from $|\vec{x}\rangle$ to $|f(\vec{x}, \vec{y})\rangle$ which are taken in superposition with amplitudes $\Phi(\vec{x}, \vec{y})$. It was shown in [1] that by restricting to *balanced* sums where $\Phi(\vec{x}, \vec{y}) = re^{iP(\vec{x}, \vec{y})}$ for some global normalization constant $r \in \mathbb{C}$, the path sum admits symbolic representation by multi-linear polynomials $P \in \mathbb{R}[\vec{X}, \vec{Y}]$ and $f = (f_1, \ldots, f_n)$, $f_i \in \mathbb{F}_2[\vec{X}, \vec{Y}]$, which is moreover polynomial-time computable from a quantum circuit or channel. In particular, for gate sets which do not include *non-linear classical* functions — a class which includes, for instance, Clifford+T and the exactly universal gate set $\{\textsc{cnot}, \textsc{r}_Z(\theta), \textsc{r}_X(\theta) := \textsc{h}\textsc{r}_Z(\theta)\textsc{h}\}$ — the number of non-zero terms in P and f remains bounded polynomially in the size of the circuit.

Example 16. Recall from Section 3 that the X, T, and H gates can be expressed as balanced path sums $\textsc{x} : |x\rangle \mapsto |1 \oplus x\rangle$, $\textsc{t} : |x\rangle \mapsto \omega |x\rangle$, and $\textsc{h} : |x\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{y \in \mathbb{F}_2} (-1)^{xy} |y\rangle$. A path integral for the circuit $U = (\textsc{hth}) \otimes \textsc{x}$ can be computed by composing the actions of each individual gate:

$$\begin{aligned}
U |x_1, x_2\rangle = (\textsc{hth} |x_1\rangle) \otimes (\textsc{x} |x_2\rangle) &= \left(\frac{1}{\sqrt{2}} \sum_{y_1} (-1)^{x_1 y_1} (\textsc{ht} |y_1\rangle) \right) \otimes |1 \oplus x_2\rangle \\
&= \left(\frac{1}{\sqrt{2}} \sum_{y_1} (-1)^{x_1 y_1} \omega^{y_1} (\textsc{h} |y_1\rangle) \right) \otimes |1 \oplus x_2\rangle \\
&= \left(\frac{1}{2} \sum_{y_1, y_2} (-1)^{x_1 y_1} \omega^{y_1} (-1)^{y_1 y_2} |y_2\rangle \right) \otimes |1 \oplus x_2\rangle \\
&= \frac{1}{2} \sum_{y_1, y_2} (-1)^{x_1 y_1 + y_2 y_1} \omega^{y_1} |y_2, 1 \oplus x_2\rangle \, .
\end{aligned}$$

We use the notation $(\!|\Psi|\!)$ to denote a path sum representation of Ψ and $(\!|\Psi|\!) \fatsemi (\!|\Psi|\!)'$ to denote the composition of path sums $(\!|\Psi|\!)$, $(\!|\Psi'|\!)$ with compatible dimensions as above. We also drop domains for variables in summations, as all variables are $\mathbb{F}_2$-valued.

A sound rewriting theory for balanced sums was further given in [1]. The rewrite system of [1] is presented in Figure 12, where rewrite rules are applied to the right-hand side of an expression $|\vec{x}\rangle \mapsto \sum_{\vec{y}} e^{iP(\vec{x}, \vec{y})} |f(\vec{x}, \vec{y})\rangle$. rewriting in the sum-over-paths can be interpreted as *contracting interfering paths*. For instance, the (H) rule of Figure 12 arises from the fact that in the path integral

$$\sum_{\vec{x}, y, z} (-1)^{y(z \oplus P(\vec{x}))} e^{iQ(\vec{x}, z)} |f(\vec{x}, z)\rangle = \sum_{\vec{x}, z} \left(\sum_{y} (-1)^{y(z \oplus P(\vec{x}))} \right) e^{iQ(\vec{x}, z)} |f(\vec{x}, z)\rangle \, ,$$

the $y = 0$ and $y = 1$ paths have opposite phase and hence cancel whenever $z \oplus P(\vec{x}) = 0$. Hence we can safely assume $z = P(\vec{x})$ in any paths with non-zero amplitude. We say that this rule is a

$$\sum_{\vec{x},y} e^{iQ(\vec{x})} |f(\vec{x})\rangle \longrightarrow_{\text{Cliff}} 2 \sum_{\vec{x}} e^{iQ(\vec{x})} |f(\vec{x})\rangle \tag{E}$$

$$\sum_{\vec{x},y,z} (-1)^{y(z\oplus P(\vec{x}))} e^{iQ(\vec{x},z)} |f(\vec{x},z)\rangle \longrightarrow_{\text{Cliff}} 2 \sum_{\vec{x}} e^{iQ(\vec{x},\overline{P}(\vec{x})} |f(\vec{x},P(\vec{x}))\rangle \tag{H}$$

$$\sum_{\vec{x},y} i^y (-1)^{yP(\vec{x})} e^{iQ(\vec{x})} |f(\vec{x})\rangle \longrightarrow_{\text{Cliff}} \omega\sqrt{2} \sum_{\vec{x}} (-i)^{\overline{P}(\vec{x})} e^{iQ(\vec{x})} |f(\vec{x})\rangle \tag{ω}$$

Fig. 12. The rewrite rules of [1]. In all rules above P is a Boolean polynomial and $z, y \notin FV(P)$ and $\overline{P}$ lifts P to an equivalent polynomial over $\mathbb{R}$ via the sound translation $\overline{P \oplus Q} = \overline{P} + \overline{Q} - 2\overline{PQ}$.

contraction on y and witnesses the constraint $z = P(\vec{x})$. More generally, given a sum of the form

$$\sum_{\vec{x},y} (-1)^{yP(\vec{x})} e^{iQ(\vec{x})} |f(\vec{x})\rangle = \sum_{\vec{x}} \left(\sum_{y} (-1)^{yP(\vec{x})} \right) e^{iQ(\vec{x})} |f(\vec{x})\rangle ,$$

we see that any path where $P(\vec{x}) \neq 0$ has zero amplitude, as the two paths corresponding to $y = 0$ and $y = 1$ again have opposite phase and destructively interfere. We again say that contraction on y witnesses the constraint $P(\vec{x}) = 0$, though we can not in general rewrite the integral as $P(\vec{x}) = 0$ may not be solvable by substitution. The (H) rule arises as a particular instance of this binary interference scheme where $P(\vec{x}) = 0$ admits a solution of the form $z = P'(\vec{x})$ where z is bound by a summation.

We say an instance of (H) is *affine* if P is some affine expression in its free variables — for instance, $P(\vec{x}) = 1 \oplus x_1 \oplus x_2 \oplus x_5$ — and more generally that it has degree k if $\deg(P) = k$. While the rewriting system of Figure 12 is not complete nor even confluent for circuits over Clifford+T, it is complete for Clifford circuits even when restricting (H) to affine instances [55].

6.2 Generating Precise Transition Relations with the Sum-Over-Paths

We now describe how path sums can be used to generate sound abstractions of the classical semantics $\mathcal{P}(C_Q) \to \mathcal{P}(C_Q)$ of a circuit, and how the rewrite rules of Figure 12 can be used to increase the precision of the abstraction.

As noted above, the sum-over-paths encodes a linear operator as a set of classical transitions

$$|\vec{x}\rangle \mapsto \{|f(\vec{x},\vec{y})\rangle \mid \vec{y} \in \mathbb{F}_2^k\}.$$

for some vector $f = (f_1, \ldots, f_n)$ of polynomials f_i. In particular, given a path sum expression $\Psi : |\vec{x}\rangle \mapsto \sum_{\vec{y}\in\mathbb{F}_2^k} \Phi(\vec{x},\vec{y}) |f(\vec{x},\vec{y})\rangle$, there exists $\vec{y} \in F_2^k$ such that the relation $\vec{x}' = f(\vec{x},\vec{y})$ holds whenever $\langle\vec{x}'| \Psi |\vec{x}\rangle \neq 0$. Hence, a sound abstraction of Ψ in the **Pol** domain is the polynomial ideal

$$\exists\vec{Y}.\langle\vec{X}' \oplus f(\vec{X},\vec{Y})\rangle = \exists\vec{Y}.\langle X_1' \oplus f_1(\vec{X},\vec{Y}), \ldots, X_n' \oplus f_n(\vec{X},\vec{Y})\rangle.$$

Definition 17. Given a path sum $(\!|\Psi|\!) = |\vec{x}\rangle \mapsto \sum_{\vec{y}\in\mathbb{F}_2^k} \Phi(\vec{x},\vec{y}) |f(\vec{x},\vec{y})\rangle$, the abstraction of $(\!|\Psi|\!)$ in the **Pol** domain is

$$\alpha\left((\!|\Psi|\!)\right) := \exists\vec{Y}.\langle\vec{X}' \oplus f(\vec{X},\vec{Y})\rangle$$

Proposition 18. *For any path sum representation $(\!|\Psi|\!)$, $\alpha\left((\!|\Psi|\!)\right)$ is a sound abstraction of Ψ.*

Proposition 19. *Let $(\!|\Psi|\!)$ and $(\!|\Psi'|\!)$ be (composable) path sums. Then*

$$\alpha((\!|\Psi|\!) \fatsemi (\!|\Psi|\!)') \sqsubseteq \alpha((\!|\Psi|\!)) \fatsemi \alpha((\!|\Psi'|\!)).$$

Note that in the case when $\deg(f_i) \leq 1$ for all i, then the polynomial mapping is in fact an affine map $(A, \vec{c})$, and can be likewise soundly abstracted in a domain of affine relations as $\vec{X}' \oplus A(\vec{X}, \vec{Y}) \oplus \vec{c}$. The affine transition relations for gates in Figure 7 arise from the specifications of each gate as a path sum in this way. If $\deg f_i > 1$, we can always soundly abstract the transition $X_i' = f_i(\vec{X}, \vec{Y})$ in an affine relational domain simply as $\top_i$.

Given a path sum $(\!|\Psi|\!)$, if $(\!|\Psi|\!) \longrightarrow_{\text{Cliff}} (\!|\Psi|\!)'$, then by the soundness of $\longrightarrow_{\text{Cliff}}$ both represent the same linear operator Ψ. In particular, $\alpha((\!|\Psi|\!)')$ gives a sound abstraction of Ψ, which may be *more precise* than $\alpha((\!|\Psi|\!))$ as the next example shows.

Example 20. Recall that the H is self-inverse, and in particular HH = I, which leads to imprecision in the phase folding analysis since $\mathcal{A}[\![\text{HH}]\!] = \mathcal{A}[\![\text{H}]\!] \fatsemi \mathcal{A}[\![\text{H}]\!] = \top$. If we compute the sum-over-paths representation of HH by composition, we see that

$$(\!|\text{H}|\!) \fatsemi (\!|\text{H}|\!) = |x\rangle \mapsto \frac{1}{2} \sum_{y,z} (-1)^{xy+yz} |z\rangle \,.$$

Noting that $\sum_{y,z}(-1)^{xy+yz}|z\rangle = \sum_{y,z}(-1)^{y(z\oplus x)}|z\rangle \longrightarrow_{\text{Cliff}} 2|x\rangle$ by (H), rewriting the above expression gives the precise classical semantics $\text{HH} : |x\rangle \mapsto |x\rangle$.

We show next that rewriting a path sum can only *increase* the precision of the abstraction, as it results from *eliminating* some interfering paths:

Proposition 21. *Let $(\!|\Psi|\!)$ by a symbolic path integral and suppose $(\!|\Psi|\!) \longrightarrow_{\text{Cliff}} (\!|\Psi|\!)'$. Then*

$$\alpha((\!|\Psi|\!)') \sqsubseteq \alpha((\!|\Psi|\!)).$$

Proof. For the (E) and (ω) rules the proof is trivial as the output expression is unchanged in either case. For the (H) rule it can be observed that

$$\alpha\left(|\vec{x}\rangle \mapsto \sum_{\vec{y},z,z'} (-1)^{z'(z\oplus P(\vec{x},\vec{y}))} e^{iQ(\vec{x},\vec{y},z)} |f(\vec{x},\vec{y},z)\rangle\right) = \exists\{\vec{Y}, Z, Z'\}.\langle \vec{X}' \oplus f(\vec{X},\vec{Y},Z)\rangle$$
$$\sqsupseteq \exists \vec{Y}.\langle \vec{X}' \oplus f(\vec{X},\vec{Y},P(\vec{X}))\rangle$$

□

Given a circuit or control-flow path π in a quantum WHILE program, by Propositions 18, 19 and 21 we can synthesize an abstraction of π which is *at least as precise* as (and often more precise than) the methods of Sections 4 and 5 by computing a path sum representation $(\!|\pi|\!)$ of π through standard techniques (e.g. [1]), reducing it using Figure 12, then abstracting it into a transition formula using a relevant domain (e.g. **Pol**). We summarize this process in Algorithm 1 below:

Algorithm 1 Synthesis of abstract transformers.

function SYNTHESIZE-TRANSFORMER(π)
  Compute a path sum $(\!|\pi|\!)$ expression of π
  while $(\!|\pi|\!) \longrightarrow_{\text{Cliff}} (\!|\pi|\!)'$, set $(\!|\pi|\!) := (\!|\pi|\!)'$
  return $\alpha((\!|\pi|\!))$
end function

Algorithm 1 can be used to generate (relatively) precise relational abstractions of basic blocks in a quantum WHILE program. As the infinite sum of path integrals however is not bounded, a suitable, terminating representation of an entire program in terms of path sums is not easily possible.

Complexity. While the *computation* of the path integral along a control-flow path is polynomial time in the number of qubits, *reduction of the path integral may in fact take exponential time* as the degree of the constituent polynomials may increase during reduction. If however rewriting is restricted to *affine* relations, Algorithm 1 terminates in time $O(m^c|\pi|)$ where $|\pi|$ is the length of the path, $m \leq n + |\pi|$ is the number of variables in the path sum, and c is the maximum degree of polynomials appearing in the path sum. For Clifford+T circuits, $c = 3$ [1].

6.3 Phase Folding Modulo Path Integral Rewriting

Algorithm 1 can be used in phase folding to generate effective summaries of the classical semantics for basic blocks. However, as in Section 4 we also wish to know which phase gates *inside* the block can be merged.

Algorithm 2 presents the Strengthen algorithm, which gathers the constraints witnessed through rewriting of the block's path integral, and then uses them to further reduce the phase conditions within the block. Strengthen attempts to construct as large of a transition ideal as possible over the pre-, post-, and intermediate-variables in the block by repeatedly adding constraints $P(\vec{x}) = 0$ witnessed by the hypothetical contraction of $\sum_{\vec{x},y}(-1)^{yP(\vec{x})}e^{iQ(\vec{x})}\,|f(\vec{x})\rangle$ on y, before picking a specific contraction to apply (i.e. applying a rewrite rule).

Algorithm 2 Strengthening phase folding by symbolic interference analysis.

function Strengthen(π, $(t, A) := \mathbf{PF}[\![\pi]\!]$)
  Compute a path sum $(\!|\pi|\!)$ expression of π
  while $(\!|\pi|\!) \longrightarrow_{\text{Cliff}} (\!|\pi|\!)'$ **do**
    Set $A := A \sqcap \langle P(\vec{X}) = 0\rangle$ whenever $(\!|\pi|\!)$ has the form $\sum_{\vec{x},y}(-1)^{yP(\vec{x})}e^{iQ(\vec{x})}\,|f(\vec{x})\rangle$
    $(\!|\pi|\!) := (\!|\pi|\!)'$
  end while
  $t := \uplus_{(f,S)\in t}\,(\text{reduce}(A, f), S)$
  return (t, A)
end function

Example 22. Consider the circuit $T := \mathrm{T}^\dagger\mathrm{HHT}$. Denoting by x, x' the pre- and post-states, and y the output of the first Hadamard gate, the phase folding analysis with either domain (affine or polynomial) returns

$$\mathbf{PF}[\![T]\!] = (t, A) = ([x \mapsto \{\ell\}, x' \mapsto \{\ell'\}], \top)\,.$$

where $x \mapsto \{\ell\}$ denotes the pair $(x, \{\ell\})$. In particular, the analysis shows that the T gate at location ℓ rotates the phase if $x = 1$, and the $\mathrm{T}^\dagger$ at location ℓ' rotates the phase if $x' = 1$, and no relation between the pre- and post-state holds. Hence, neither gate can be merged with the other, despite the circuit being equal to the identity. Computing the path sum representation of T however, we see that

$$T : |x\rangle \mapsto \frac{1}{2}\sum_{y,x'}(-1)^{xy+yx'}\omega^{x-x'}\,|x'\rangle\,.$$

The sum can be contracted on y, which witnesses the equality $x' = x$:

$$\frac{1}{2}\sum_{y,x'}(-1)^{xy+yx'}\omega^{x-x'}\,|x'\rangle = \sum_{y,x'}(-1)^{y(x\oplus x')}\omega^{x-x'}\,|x'\rangle \xrightarrow{\text{(H)}}_{\text{Cliff}} \omega^{x-x}\,|x\rangle = |x\rangle\,.$$

The Strengthen algorithm uses this equality to strengthen the transition ideal by setting $A := \top \sqcap \langle x' \oplus x\rangle = \langle x' \oplus x\rangle$. Finally, Strengthen reduces all phase conditions modulo A, where in

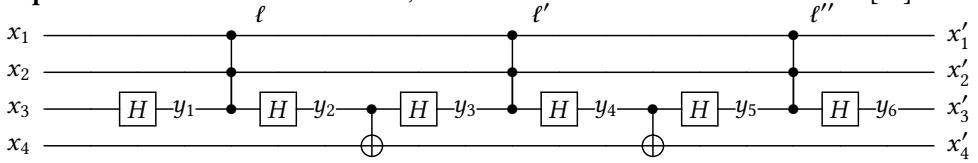
particular $\text{reduce}(A, x) = x = \text{reduce}(A, x')$ and the algorithm hence returns the stronger result $([x \mapsto \{\ell, \ell'\}], \langle x = x' \rangle)$ which shows the T and $T^\dagger$ gate can be merged.

In the above example, the optimization could be achieved in a forward-manner similar to Section 4.4. In particular, after the second Hadamard gate the path integral is

$$|x\rangle \mapsto \frac{1}{2} \sum_{y,x'} (-1)^{xy+yx'} \omega^x |x'\rangle = |x\rangle$$

which reduces via (H) to $|x\rangle$, at which point applying the final $T^\dagger$ to $|x\rangle$ will merge with the first T. Strengthen works instead by first computing a path integral for the *entire block*, and then applying path sum reductions to strengthen the analysis. This is for technical reasons owing to the fact that the rewriting system of Figure 12 is non-confluent in general. The example below illustrates this fact and its impact on phase folding.

Example 23. Consider the circuit below, which occurs in the 8-bit adder circuit of [50].



Intermediate variables corresponding to the outputs of Hadamard gates are labeled in the circuit for convenience. Recall that the CCZ gates depicted as ⋮ implement the diagonal transformation $\text{CCZ} : |x_1, x_2, x_3\rangle \mapsto (-1)^{x_1 x_2 x_3} |x_1, x_2, x_3\rangle$. The result of the phase folding analysis, extended to this gate set, is a pair (t, A) where

$$t = [x_1x_2y_1 \mapsto \{\ell\}, x_1x_2y_3 \mapsto \{\ell'\}, x_1x_2y_5 \mapsto \{\ell''\}],$$
$$A = \langle x'_1 = x_1, x'_2 = x_2, x'_3 = y_6, x'_4 = x_4 \oplus y_2 \oplus y_4 \rangle.$$

Computing the circuit's path integral, we similarly get

$$|x_1, x_2, x_3, x_4\rangle \mapsto \sum_{\vec{y} \in \mathbb{F}_2^6} (-1)^{y_1(x_3 \oplus x_1x_2 \oplus y_2) + y_3(y_2 \oplus x_1x_2 \oplus y_4) + y_5(y_4 \oplus x_1x_2 \oplus y_6)} |x_1, x_2, y_6, x_4 \oplus y_2 \oplus y_4\rangle,$$

where the phase polynomial is factored to show the three possible contractions on y_1, y_3, and y_5, witnessing the constraints $x_3 \oplus x_1x_2 \oplus y_2 = 0$, $y_2 \oplus x_1x_2 \oplus y_4 = 0$, and $y_4 \oplus x_1x_2 \oplus y_6 = 0$, respectively. Strengthen hence computes

$$A' := A \sqcap \langle x_3 \oplus x_1x_2 \oplus y_2, y_2 \oplus x_1x_2 \oplus y_4, y_4 \oplus x_1x_2 \oplus y_6 \rangle.$$

Applying the contraction on y_1 and substituting $y_2 \leftarrow x_1x_2 \oplus x_3$ then yields

$$\sum (-1)^{y_3(x_3 \oplus y_4) + y_5(y_4 \oplus x_1x_2 \oplus y_6)} |x_1, x_2, y_6, x_4 \oplus x_3 \oplus x_1x_2 \oplus y_4\rangle.$$

Contraction on either y_3 or y_5 yields no new equations, as both $x_3 \oplus y_4 \in A'$ and $y_4 \oplus x_1x_2 \oplus y_6 \in A'$. Picking y_3 to contract on next and substituting $y_4 \leftarrow x_3$ yields

$$\sum (-1)^{y_5(x_3 \oplus x_1x_2 \oplus y_6)} |x_1, x_2, y_6, x_4 \oplus x_1x_2\rangle.$$

Again, $x_3 \oplus x_1x_2 \oplus y_6 \in A'$ and the final contraction on y_5 produces the precise transition relation $|x_1, x_2, x_3, x_4\rangle \mapsto |x_1, x_2, x_3 \oplus x_1x_2, x_4 \oplus x_1x_2\rangle$. The final step is to reduce t with respect to A', which we can do by computing a reduced Gröbner basis (in grevlex order)

$$A' = \langle x_3 \oplus y_2 \oplus x_1x_2, x_3 \oplus y_2 \oplus x_2x_3 \oplus x_2y_2, x_3 \oplus y_2 \oplus x_1x_3 \oplus x_1y_2, x_3 \oplus y_4, y_2 \oplus y_6 \rangle.$$

Reducing t modulo A' then gives $[x_3y_1 \oplus y_1y_2 \mapsto \{\ell\}, x_3y_3 \oplus y_2y_3 \mapsto \{\ell'\}, x_3y_5 \oplus y_2y_5 \mapsto \{\ell''\}]$ and hence no gates can be merged.

If however the first contraction chosen was n y_3, we get the following path integral

$$\sum (-1)^{y_1x_3+y_4(y_1\oplus y_5)+y_5x_1x_2+y_5y_6} |x_1, x_2, y_6, x_4 \oplus x_1x_2\rangle .$$

We now see that interference on y_4 witnesses the constraint $y_1 \oplus y_5 = 0$. Setting $A'' := A' \sqcap \langle y_1 \oplus y_5 \rangle$ and reducing t modulo A'' we get

$$[x_3y_3 \oplus y_2y_3 \mapsto \{\ell'\}, x_3y_1 \oplus y_1y_2 \mapsto \{\ell, \ell''\}]$$

which in particular allows the elimination of the ccz gates at locations ℓ and ℓ''.

It remains a direction for future work to determine methods of both confluently rewriting the path sum, as well as to efficiently find all polynomial constraints witnessed by binary interference of the form $\sum_y (-1)^{yP(\vec{x})}$.

Implementation. In practice, Algorithm 2 generates ideals which are too large for our implementation to feasibly compute reduced Gröbner bases, as the number of intermediate variables is proportional to the length of the circuit block. Instead, we non-canonically reduce phase conditions with respect to the full set of generated equations, and hence our implementation theoretically misses some gates which can be merged with Algorithm 2. For small programs in which Gröbner bases could be computed, we saw no difference in optimization between the two methods. To speed up reduction we also experimented with methods of *linearizing* the ideal and reducing via Gaussian elimination, but saw little gain in performance.

7 Experimental Evaluation

We implemented our optimizations in the open-source package Feynman[2]. The sources and benchmarks used in the experiments are provided in the associated software artifact [4]. We denote the affine optimization of Section 4 by $\mathbf{PF}_{\text{Aff}}$, and the polynomial optimization of Section 5 combined with the Strengthen algorithm of Section 6 by $\mathbf{PF}_{\text{Pol}}$. We also ran experiments where Strengthen was limited to *quadratic* relations, which we denote by $\mathbf{PF}_{\text{Quad}}$, in order to ascertain the effectiveness of highly non-linear relations over those which characterize the Toffoli gate. In both the unbounded and quadratic cases, reduction in Strengthen was performed using non-canonical multivariate division, as computing a full Gröbner basis caused almost all benchmarks to time out.

We performed two sets of experiments: *program* optimization experiments which involve classical control, and straightline *circuit* optimization experiments. All experiments were run in Ubuntu 22.04 running on an AMD Ryzen 5 5600G 3.9GHz processor and 64GB of RAM. Optimizations were given a 2-hour TIMEOUT and 32GB memory limit.

7.1 Program Optimization Benchmarks

For the program optimization benchmarks, a front-end for the quantum programming language openQASM 3 [13] was written and used to perform optimizations on quantum programs which combine circuits and classical control. A subset of openQASM 3 which restricts loops to a single entry and exit point is modeled as a non-deterministic quantum WHILE program by (1) eliminating any classical computation, (2) replacing branch and loop conditions with $\star$, and (3) inlining *gate* calls. Note that procedure calls are *not* inlined whereas gate calls are, as most optimizations in practice occur across sub-circuit boundaries.

The results for our program optimization benchmarks are present in Table 1. As we could not find a set of suitable optimization benchmarks in openQASM 3, we tested our relational optimization on a selection of hand-written micro-benchmarks designed to test the ability to compute certain

[2] github.com/meamy/feynman

Table 1. Optimization of T-count in hybrid openQASM 3.0 micro-benchmarks. All other non-diagonal gate counts are left unchanged. The loop invariant computed by $\mathbf{PF}_{\text{Pol}}$ is also given.

Benchmark	n	Original	$\mathbf{PF}_{\text{Aff}}$		$\mathbf{PF}_{\text{Pol}}$		Loop invariant
		# T	T count	time (s)	# T	time (s)	
RUS	3	16	10	0.30	8	0.35	$\langle z' \oplus z \rangle$
Grover	129	1736×10^9	1470×10^9	1.98	TIMEOUT		–
Reset-simple	2	2	1	0.15	1	0.23	–
If-simple	2	2	0	0.18	0	0.16	–
Loop-simple	2	2	0	0.17	0	0.16	$\langle x' \oplus x, y \oplus y' \oplus xy \oplus xy' \rangle$
Loop-h	2	2	0	0.16	0	0.16	$\langle y' \oplus y \rangle$
Loop-nested	2	3	2	0.17	2	0.18	$\langle x' \oplus x \rangle, \langle x' \oplus x \rangle$
Loop-swap	2	2	0	0.30	0	0.20	$\langle x' \oplus y' \oplus x \oplus y, x' \oplus xy \oplus xx' \oplus yx' \rangle$
Loop-nonlinear	3	30	18	0.44	0	0.26	$\langle x' \oplus x, z' \oplus z, y' \oplus y \oplus xy \oplus xy' \rangle$
Loop-null	2	4	1	0.18	1	0.17	$\langle x' \oplus x, y' \oplus y \rangle$

invariants and to perform the associated optimizations. The benchmarks are provided in the Feynman repository[3], and include the examples given throughout the paper – notably Figure 2 (RUS), Figure 3 (Loop-swap), and Figure 11 (Loop-nonlinear). For the Grover benchmark, an instance of Grover's search on a 64-bit function was generated and the loop ($\approx 10^9$ iterations) was modeled as a non-deterministic loop for optimization. The results in Table 1 confirm that our methods are able to find some non-trivial optimizations in simple hybrid programs which have interesting classical control. Moreover, the experiments demonstrate that phase folding can be integrated into compilers targeting hybrid quantum/classical programming languages.

7.2 Circuit Optimization Benchmarks

To test the impacts of our relational approach and the use of non-linear (classical) reasoning through the Strengthen algorithm to optimize T-counts, we also performed circuit optimization experiments using a standard set of circuit benchmarks [35]. We performed two sets of experiments — ones to isolate the impact of Strengthen on phase folding, and ones to evaluate the overall effectiveness of our optimizations against other (non-monotone) circuit optimization algorithms. For the isolation experiments, we compared against PyZX [34] which implements a variant of phase folding in the ZX-calculus [35]. We chose this version of the phase folding algorithm as it combines phase folding with Clifford normalization, and in particular reaches the theoretical limits of phase folding up to equalities and commutations of Clifford circuits [53]. As Clifford computations implement *affine* classical state transitions, our hypothesis is that our affine optimization algorithm $\mathbf{PF}_{\text{Aff}}$ should match PyZX's optimizations in all cases. To test the overall efficacy of our optimizations in reducing T-count, we also compared our optimizations against a selection of circuit optimizers (VOQC [28], PyZX [34], FastTODD [54], QUESO [56]) which implement variants of phase folding (PyZX, VOQC), peephole-optimizations (VOQC, QUESO), and optimizations based on Reed-Muller decoding (FastTODD). As Reed-Muller optimizers apply an intrinsically different class of optimizations than phase folding [46], we also performed experiments comparing FastTODD after first either applying PyZX, or applying $\mathbf{PF}_{\text{Pol}}$. All circuits optimized by Feynman were validated for correctness using the method of [1].

Table 2 presents a summary of our experimental results, showing the isolation experiments and the overall best T-count optimization results. The full experimental results are provided as supplementary material in the ACM digital library. As noted in [56], peephole optimizations are generally ineffective at reducing T-count as they are highly local. This is reflected in our results which show that either $\mathbf{PF}_{\text{Pol}}$, PyZX+FastTODD, or $\mathbf{PF}_{\text{Pol}}$+FastTODD always outperform

[3] github.com/meamy/feynman/tree/ara/benchmarks/qasm3

Table 2. T-count optimization evaluation. For PyZX and **PF** results, all other non-diagonal gate counts are left unchanged. Bolded entries in the **PF** columns denote entries which outperform PyZX, and the best entry (if it exists) in the +FastTODD columns is also bolded.

Benchmark	n	Original	PyZX		$\mathbf{PF}_{\text{Aff}}$		$\mathbf{PF}_{\text{Quad}}$		$\mathbf{PF}_{\text{Pol}}$		+FastTODD	
		# T	# T	time (s)	# T	time (s)	# T	time (s)	# T	time (s)	PyZX	$\mathbf{PF}_{\text{Pol}}$
Grover_5	9	336	166	0.26	**148**	0.18	**148**	0.2	**0**	0.47	143	**0**
Mod 5_4	5	28	8	0.01	8	<0.01	8	<0.01	8	0.47	7	7
VBE-Adder_3	10	70	24	0.02	24	0.01	24	0.01	24	0.01	19	19
CSLA-MUX_3	15	70	62	0.03	**60**	0.01	**60**	0.01	**60**	0.02	39	39
CSUM-MUX_9	30	196	84	0.08	84	0.02	84	0.02	84	0.15	71	71
QCLA-Com_7	24	203	95	0.12	**94**	0.04	**94**	0.04	**94**	0.5	**59**	61
QCLA-Mod_7	26	413	237	0.38	237	0.17	237	0.19	237	39.13	**159**	161
QCLA-Adder_10	36	238	162	0.11	162	0.05	162	0.06	162	0.85	109	109
Adder_8	24	399	173	0.51	173	0.18	173	0.21	173	2.31	**119**	121
RC-Adder_6	14	77	47	0.04	47	0.02	47	0.02	47	0.05	37	37
Mod-Red_21	11	119	73	0.06	73	0.02	73	0.02	73	0.04	51	51
Mod-Mult_55	9	49	35	0.02	35	0.01	35	0.01	35	0.01	17	17
Mod-Adder_1024	28	1995	1011	3	1011	14.01	**1005**	14.16	**923**	36.34	–	–
GF(2^4)-Mult	12	112	68	0.06	68	0.02	68	0.02	68	0.02	49	49
GF(2^5)-Mult	15	175	115	0.08	115	0.03	115	0.04	115	0.04	81	**75**
GF(2^6)-Mult	18	252	150	0.13	150	0.07	150	0.07	150	0.04	113	**111**
GF(2^7)-Mult	21	343	217	0.25	217	0.11	217	0.11	217	0.09	155	**141**
GF(2^8)-Mult	27	448	264	0.63	264	0.4	264	0.4	264	0.46	205	**203**
GF(2^9)-Mult	24	567	351	0.5	351	0.37	351	0.38	351	0.44	257	**255**
GF(2^{10})-Mult	27	700	410	1.06	410	0.61	410	0.6	410	0.7	315	**313**
GF(2^{16})-Mult	48	1792	1040	5.5	1040	17.05	1040	16.87	1040	18.28	**797**	803
GF(2^{32})-Mult	96	7168	4128	9.33m	4128	67m	4128	60m	4128	86m	–	–
Ham_15 (low)	17	161	97	0.46	97	0.31	97	0.3	97	0.44	77	77
Ham_15 (med)	17	574	212	1.8	212	0.3	212	0.31	**210**	1.59	**137**	140
Ham_15 (high)	20	2457	1019	35.09	1019	21.33	**997**	21.23	**985**	2m	–	–
HWB_6	7	105	75	0.07	75	0.02	75	0.03	75	0.12	51	51
QFT_4	5	69	67	0.02	67	0.01	**65**	0.01	**65**	0.03	**53**	54
$\Lambda_3(X)$	5	21	15	0.01	15	<0.01	15	<0.01	15	<0.01	13	13
$\Lambda_4(X)$	7	35	23	0.01	23	<0.01	23	<0.01	23	<0.01	19	19
$\Lambda_5(X)$	9	49	31	0.01	31	<0.01	31	0.01	31	0.01	25	25
$\Lambda_{10}(X)$	19	119	71	0.04	71	0.01	71	0.02	71	0.01	55	55
$\Lambda_3(X)$ (dirty)	5	28	16	0.01	16	0.01	16	0.01	16	0.01	13	13
$\Lambda_4(X)$ (dirty)	7	56	28	0.02	28	0.01	**24**	0.01	**24**	0.01	23	**20**
$\Lambda_5(X)$ (dirty)	9	84	40	0.03	40	0.01	**32**	0.01	**32**	0.01	33	**28**
$\Lambda_{10}(X)$ (dirty)	19	224	100	0.11	100	0.05	**72**	0.03	**72**	0.06	83	**68**
FP-renorm	10	112	94	0.05	**81**	0.05	**81**	1.34	**71**	2.65	69	**56**

the peephole optimizers, even when phase folding is also applied as a pre-processing step to QUESO. The peephole optimizers however often produce the best total gate counts, with FastTODD producing the worst overall gate counts due to a process of *gadgetization* and re-synthesis. Overall, our results show that combining $\mathbf{PF}_{\text{Pol}}$ with Reed-Muller optimizations produces on average the best T counts, and usually outperforms PyZX+FastTODD when $\mathbf{PF}_{\text{Pol}}$ outperforms PyZX. We suspect cases where $\mathbf{PF}_{\text{Pol}}$ did not improve on PyZX but PyZX+FastTODD outperformed $\mathbf{PF}_{\text{Pol}}$+FastTODD to be due to non-determinism in the FastTODD optimization.

Our isolation experiments confirm that *affine* phase folding is able to match the T gate counts obtained by PyZX in every case. Thew few instances in which $\mathbf{PF}_{\text{Aff}}$ outperforms PyZX are due to Feynman's use of $|0\rangle$-initialized ancillas to further optimize T-counts, which PyZX does not do. Our experiments further confirm that by adding in *non-linear* equalities, phase folding is able to optimize away more T gates in many cases. We detail some of these cases below.

One such class of circuits where $\mathbf{PF}_{\text{Pol}}$ consistently outperforms PyZX and $\mathbf{PF}_{\text{Aff}}$ are the $\Lambda_k(X)$ benchmarks with *dirty ancillas*, corresponding to an implementation due to [7]. The $\Lambda_4(X)$ benchmark is reproduced in Figure 13. Observe that the equality $a' = a$ can be used to optimize away 4 T gates from the indicated Toffolis. Discovering the equality via generic means requires non-linear

reasoning, as it relies on the intermediate equalities $a'' = a \oplus xy$ and $a' = a'' \oplus xy$, and hence has previously been achieved by hand-optimization [37]. Our results match the T-count scaling of $8(k-1)$ for the hand-optimized $\Lambda_k(X)$ implementation from [37] — to the best of our knowledge, ours is the first circuit optimization to reach this T-count via automated means.

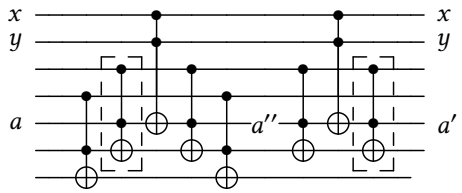


Fig. 13. Implementation of the 4-control Toffoli gate with 2 dirty ancillas.

Benchmarks where $\mathbf{PF}_{\text{Pol}}$ outperformed both affine and *quadratic* phase folding include Grover_5 and FP-renorm. In the former case, $\mathbf{PF}_{\text{Pol}}$ is able to reduce the T-count to 0, owing to the fact that rewriting the circuit's path integral gives the trivial transition $|00\cdots0\rangle \mapsto \frac{1}{8}\sum_{\vec{y}\in\mathbb{F}_2^6}(-1)^{y_0}|\vec{y}, 000\rangle$. It appears this is an error in the implementation of Grover's algorithm. The FP-renorm benchmark, taken from [26], was added specifically to provide a separation between quadratic and higher-degree optimizations. The circuit implements renormalization of the mantissa in floating point calculations, and it was shown that the effective T-count can be reduced to 70 by using a (programmer-supplied) state invariant which expresses a linear inequality over $\mathbb{Z}_k$. As expected, our results show a separation between T-count optimizations possible with only quadratic equations (81) and those possible using higher-degree equations (71), which are generally needed when reasoning about integer arithmetic at the bit-level.

8 Conclusion

In this paper we have described a generalization of the quantum phase folding circuit optimization to quantum *programs*. Our algorithm works by performing an relational analysis approximating the classical transitions of the program in order to determine where phase gates can be merged. We gave two concrete domains, one for affine relations based on [18], and the other for polynomial relations based on Gröbner basis methods. We further showed that abstract transformers for these domains can be generated by rewriting the circuit sum-over-paths. Our experimental results show that relational analysis is able to go improve upon existing circuit optimizations in many cases as well as discover some non-trivial optimizations of looping programs.

Many open questions and avenues for future work remain, particular generation of the *most precise* transition relation for a quantum circuit. Other directions include expanding features of the analysis to include *classical* logic such as branch conditions, and to experiment with the many existing classical techniques for relational analysis. A promising direction motivated by our experimental results is to explore techniques for *degree-bounded* polynomial relations, for example [38]. More broadly, this work raises the question of how else might *classical* program analysis techniques — particularly, numerical ones — be applied in a quantum setting? One such avenue is to instead consider numerical invariants over the $\mathbb{Z}_N$, corresponding to the *modular* basis $\{|i\rangle \mid i \in \mathbb{Z}_{2^n}\}$ of n qubits which is frequently used in the analysis of quantum algorithms like Shor's seminal algorithm [48]. Yet another direction is to consider how the types of invariants we generate here may be applied to the *formal verification* of quantum programs.

8.1 Related Work

Quantum circuit optimization. The concept of *phase folding* originated in the phase-polynomial optimization of [5] and was later generalized from a re-synthesis algorithm to an in-place optimization in [40]. The optimization was later re-formulated [35, 59] in the Pauli exponential and ZX-calculus frameworks, which extended the optimization to perform equivalent optimizations

over all Pauli bases. Both methods can be seen as phase folding "up to Clifford equalities", which is reflected in their experimental validation which curiously produced identical T-counts across all benchmarks. Our method of rewriting the path integral, when restricted to *affine* relations, can be seen as performing the same process of Clifford normalization, a fact which is mirrored in our experimental results for $\mathbf{PF}_{\text{Aff}}$ which coincide with [35, 59]. By using the symbolic nature of path integrals, we extend optimization further to *non-Clifford* equivalences.

If the problem of phase folding is relaxed to include *re-synthesis*, another avenue of phase gate reduction opens up — notably, techniques based on Reed-Muller codes [6]. Many recent phase gate optimization methods [16, 27, 46] make use of these techniques to often achieve lower T-counts than those reported here. However, these methods rely on *gadgetization* whereby $O(|C|)$ ancillas are added to the circuit C, and are generally orthogonal to phase folding [35]. We do note that even recent gadget-based work [46] was unable to find the optimizations of the dirty ancilla Toffoli gates found by our methods.

Quantum program optimization. Unlike circuit optimization, quantum *program* optimization is relatively unstudied. The work of [29] explicitly broaches the subject of dataflow optimization for quantum programs, developing a single static assignment (SSA) form for quantum programs and applying optimizations like gate cancellation on top of it. This notion differs from our notion of dataflow as their dataflow is explicitly classical, corresponding to the flow of *quantum* values through a *classical* control flow graph executing quantum operations at the nodes. In contrast, our model corresponds to *classical* values with flow that is either *either quantum (in superposition) or classical (out of superposition)*.

The work of [26] shares many similarities with our approach. In particular, they use assertions expressed on the classical support of the quantum state to perform a number of local gate elimination optimizations. While their assertions are more expressive than the invariants we generate, as they include among other properties *numerical inequalities* over the modular basis $\{|i\rangle \mid i \in \mathbb{Z}_{2^n}\}$, their assertions are user-supplied and not checked by a compiler. By contrast, our invariants are provably safe, generated automatically and as *relational* invariants apply to non-local gate optimizations.

Relational program analysis. Our work can be viewed as applying existing classical methods of relational, specifically algebraic, program analysis to the quantum domain. Our affine analysis directly uses the domain due to [33] and [18], which can be traced back to its origins in [31]. Likewise, our polynomial analysis uses the theory of *transition ideals* from the recent work [14] and the algebraic foundations of [44]. Further analogies with relational analysis can be made between our method of generating precise transition relations and the notion of computing the *best symbolic transformer* in [43], though the methods themselves share little similarities.

Quantum abstract interpretation. Abstract interpretation has previously been applied to the problems of simulating circuits [9] or verifying properties of circuits including separability [42] and projection-based assertions [21, 58]. The work of [58] gave an abstract domain of products of subspaces on subsets of qubits and applied this to the verification of assertions in static (but large) quantum circuits. Later work [21] defined a subspace domain $\mathcal{S}(\mathcal{H}_Q)$ consisting of subspaces of a Hilbert space $\mathcal{H}_Q$ and gave operators for the interpretation of quantum WHILE programs, as well as translations to and from Hoare logics based on Birkhoff von-Neumann (subspace) logic. In contrast to these works, we abstract subspaces of the *classical* states $C_Q = \mathbb{F}_2^{|Q|}$, which crucially admits a polynomial-size basis and hence remains tractable. While the local subspace domain of [58] likewise remains polynomial-size, our methods apply to general quantum programs and are able to prove the assertions for both the BV and GHZ test cases, as both circuits are precisely simulable using affine transformer semantics.

Acknowledgments

We wish to thank the anonymous POPL reviewers for their comments and suggestions which have greatly improved the presentation of this paper. MA acknowledges support from Canada's NSERC and the Canada Research Chairs program.

References

[1] Matthew Amy. 2018. Towards Large-Scale Functional Verification of Universal Quantum Circuits. In *Proceedings of the 15th International Conference on Quantum Physics and Logic (QPL '18)*. 1–21. https://doi.org/10.4204/EPTCS.287.1

[2] Matthew Amy. 2019. *Formal Methods in Quantum Circuit Design*. Ph. D. Dissertation. University of Waterloo. http://hdl.handle.net/10012/14480

[3] Matthew Amy and Vlad Gheorghiu. 2020. staq—A full-stack quantum processing toolkit. *Quantum Science and Technology* 5, 3 (Jun. 2020), 034016. https://doi.org/10.1088/2058-9565/ab9359

[4] Matthew Amy and Joseph Lunderville. 2024. *Linear and Non-linear Relational Analyses for Quantum Program Optimization: Artifact*. https://doi.org/10.5281/zenodo.13921830

[5] 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 (Oct. 2014), 1476–1489. https://doi.org/10.1109/TCAD.2014.2341953

[6] Matthew Amy and Michele Mosca. 2019. T-count optimization and Reed-Muller codes. *IEEE Transactions on Information Theory* 65, 8 (March 2019), 4771–4784. https://doi.org/10.1109/TIT.2019.2906374 arXiv:1601.07363

[7] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. 1995. Elementary gates for quantum computation. *Physical Review A* 52 (Nov. 1995), 3457–3467. Issue 5. https://doi.org/10.1103/PhysRevA.52.3457

[8] Ethan Bernstein and Umesh Vazirani. 1997. Quantum Complexity Theory. *SIAM J. Comput.* 26, 5 (1997), 1411–1473. https://doi.org/10.1137/S0097539796300921

[9] Benjamin Bichsel, Anouk Paradis, Maximilian Baader, and Martin Vechev. 2023. Abstraqt: Analysis of Quantum Circuits via Abstract Stabilizer Simulation. *Quantum* 7 (Nov. 2023), 1185. https://doi.org/10.22331/q-2023-11-20-1185

[10] Michael Brickenstein and Alexander Dreyer. 2009. PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials. *Journal of Symbolic Computation* 44, 9 (Sep. 2009), 1326–1345. https://doi.org/10.1016/j.jsc.2008.02.017

[11] B. Buchberger. 1976. A theoretical basis for the reduction of polynomials to canonical forms. *SIGSAM Bull.* 10, 3 (Aug. 1976), 19–29. https://doi.org/10.1145/1088216.1088219

[12] David A. Cox, John Little, and Donal O'Shea. 2010. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra* (3rd ed.). Springer Publishing Company, Incorporated.

[13] Andrew Cross, Ali Javadi-Abhari, Thomas Alexander, Niel De Beaudrap, Lev S. Bishop, Steven Heidel, Colm A. Ryan, Prasahnt Sivarajah, John Smolin, Jay M. Gambetta, and Blake R. Johnson. 2022. OpenQASM 3: A Broader and Deeper Quantum Assembly Language. *ACM Transactions on Quantum Computing* 3, 3 (Sep. 2022). https://doi.org/10.1145/3505636

[14] John Cyphert and Zachary Kincaid. 2024. Solvable Polynomial Ideals: The Ideal Reflection for Program Analysis. *Proceedings of the ACM on Programming Languages* 8, POPL, Article 25 (Jan. 2024), 724–752 pages. https://doi.org/10.1145/3632867

[15] Christopher M. Dawson, Andrew P. Hines, Duncan Mortimer, Henry L. Haselgrove, Michael A. Nielsen, and Tobias J. Osborne. 2005. Quantum computing and polynomial equations over the finite field $\mathbb{Z}_2$. *Quantum Information and Computation* 5, 2 (Mar. 2005), 102–112. https://doi.org/10.26421/QIC5.2-2

[16] Niel de Beaudrap, Xiaoning Bian, and Quanlong Wang. 2020. Fast and Effective Techniques for T-Count Reduction via Spider Nest Identities. In *Proceedings of the 15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC '20)*. 11:1–11:23. https://doi.org/10.4230/LIPIcs.TQC.2020.11

[17] Bryan Eastin and Emanuel Knill. 2009. Restrictions on Transversal Encoded Quantum Gate Sets. *Physical Review Letters* 102 (Mar. 2009), 110502. Issue 11. https://doi.org/10.1103/PhysRevLett.102.110502

[18] Matt Elder, Junghee Lim, Tushar Sharma, Tycho Andersen, and Thomas Reps. 2014. Abstract Domains of Affine Relations. *ACM Transactions on Programming Languages and Systems* 36, 4, Article 11 (Oct. 2014), 73 pages. https://doi.org/10.1145/2651361

[19] Jean-Charles Faugére. 1999. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra* 139, 1 (Jun. 1999), 61–88. https://doi.org/10.1016/S0022-4049(99)00005-5

[20] Yuan Feng, Runyao Duan, Zhengfeng Ji, and Mingsheng Ying. 2007. Proof rules for the correctness of quantum programs. *Theoretical Computer Science* 386, 1 (Oct. 2007), 151–166. https://doi.org/10.1016/j.tcs.2007.06.011

[21] Yuan Feng and Sanjiang Li. 2023. Abstract interpretation, Hoare logic, and incorrectness logic for quantum programs. *Information and Computation* 294 (Oct. 2023), 105077. https://doi.org/10.1016/j.ic.2023.105077
[22] Richard P. Feynman and Albert R. Hibbs. 1965. *Quantum mechanics and path integrals.* McGraw-Hill.
[23] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. 2012. Surface Codes: Towards Practical Large-scale Quantum Computation. *Physical Review A* 86 (Sep. 2012), 032324. Issue 3. https://doi.org/10.1103/PhysRevA.86.032324
[24] Sicun Gao, André Platzer, and Edmund M. Clarke. 2011. Quantifier elimination over finite fields using Gröbner bases. In *Proceedings of the 4th International Conference on Algebraic Informatics (CAI '11)*. 140–157. https://doi.org/10.1007/978-3-642-21493-6_9
[25] Daniel Gottesman and Isaac L. Chuang. 1999. Quantum Teleportation is a Universal Computational Primitive. *Nature* 402, 6760 (Nov. 1999), 390–393. https://doi.org/10.1038/46503
[26] Thomas Häner, Torsten Hoefler, and Matthias Troyer. 2020. Assertion-based optimization of Quantum programs. *Proceedings of the ACM on Programming Languages* 4, OOPSLA, Article 133 (Nov. 2020), 20 pages. https://doi.org/10.1145/3428201
[27] Luke E. Heyfron and Earl T. Campbell. 2018. An Efficient Quantum Compiler that Reduces T Count. *Quantum Science and Technology* 4, 1 (Sep. 2018), 015004. https://doi.org/10.1088/2058-9565/aad604
[28] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. 2021. A verified optimizer for Quantum circuits. *Proceedings of the ACM on Programming Languages* 5, POPL, Article 37 (Jan. 2021), 29 pages. https://doi.org/10.1145/3434318
[29] David Ittah, Thomas Häner, Vadym Kliuchnikov, and Torsten Hoefler. 2022. QIRO: A Static Single Assignment-based Quantum Program Representation for Optimization. *ACM Transactions on Quantum Computing* 3, 3, Article 14 (Jun. 2022), 32 pages. https://doi.org/10.1145/3491247
[30] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J. Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D. Nation, Lev S. Bishop, Andrew W. Cross, Blake R. Johnson, and Jay M. Gambetta. 2024. Quantum computing with Qiskit. (2024). arXiv:2405.08810 [quant-ph]
[31] Michael Karr. 1976. Affine relationships among variables of a program. *Acta Informatica* 6, 2 (Jun. 1976), 133–151. https://doi.org/10.1007/BF00268497
[32] Zachary Kincaid, Thomas Reps, and John Cyphert. 2021. Algebraic Program Analysis. In *Proceedings of the 33rd International Conference on Computer Aided Verification (CAV '21)*. 46–83. https://doi.org/10.1007/978-3-030-81685-8_3
[33] Andy King and Harald Søndergaard. 2008. Inferring Congruence Equations Using SAT. In *Proceedings of the 20th International Conference on Computer Aided Verification (CAV '08)*. 281–293. https://doi.org/10.1007/978-3-540-70545-1_26
[34] Aleks Kissinger and John van de Wetering. 2020. PyZX: Large Scale Automated Diagrammatic Reasoning. In *Proceedings of the 16th International Conference on Quantum Physics and Logic (QPL '19)*. 229–241. https://doi.org/10.4204/eptcs.318.14
[35] Aleks Kissinger and John van de Wetering. 2020. Reducing the number of non-Clifford gates in quantum circuits. *Physical Review A* 102 (Aug. 2020), 022406. Issue 2. https://doi.org/10.1103/PhysRevA.102.022406
[36] Daniel Litinski. 2019. A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery. *Quantum* 3 (Mar. 2019), 128. https://doi.org/10.22331/q-2019-03-05-128
[37] Dmitri Maslov. 2016. Advantages of using relative-phase Toffoli gates with an application to multiple control Toffoli optimization. *Physical Review A* 93 (Feb. 2016), 022311. Issue 2. https://doi.org/10.1103/PhysRevA.93.022311
[38] Markus Müller-Olm and Helmut Seidl. 2004. A Note on Karr's Algorithm. In *Automata, Languages and Programming (ICALP '04)*. 1016–1028. https://doi.org/10.1007/978-3-540-27836-8_85
[39] Markus Müller-Olm and Helmut Seidl. 2004. Precise interprocedural analysis through linear algebra. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '04)*. 330–341. https://doi.org/10.1145/964001.964029
[40] 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 (May 2018), 23. https://doi.org/10.1038/s41534-018-0072-4
[41] Michael A. Nielsen and Isaac L. Chuang. 2000. *Quantum Computation and Quantum Information.* Cambridge University Press.
[42] Simon Perdrix. 2008. Quantum Entanglement Analysis Based on Abstract Interpretation. In *Proceedings of the 15th International Symposium on Static Analysis (SAS '18)*. https://doi.org/10.1007/978-3-540-69166-2_18
[43] Thomas Reps, Mooly Sagiv, and Greta Yorsh. 2004. Symbolic Implementation of the Best Transformer. In *Verification, Model Checking, and Abstract Interpretation (VMCAI '04)*. 252–266. https://doi.org/10.1007/978-3-540-24622-0_21
[44] Enric Rodríguez-Carbonell and Deepak Kapur. 2004. Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations. In *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation (ISSAC '04)*.

266–273. https://doi.org/10.1145/1005285.1005324
[45] Martin Roetteler. 2008. Quantum algorithms for highly non-linear Boolean functions. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, 448–457. https://doi.org/10.1137/1.9781611973075.37
[46] Francisco J. R. Ruiz, Tuomas Laakkonen, Johannes Bausch, Matej Balog, Mohammadamin Barekatain, Francisco J. H. Heras, Alexander Novikov, Nathan Fitzpatrick, Bernardino Romera-Paredes, John van de Wetering, Alhussein Fawzi, Konstantinos Meichanetzidis, and Pushmeet Kohli. 2024. Quantum Circuit Optimization with AlphaTensor. (2024). arXiv:2402.14396
[47] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. 2004. Non-linear loop invariant generation using Gröbner bases. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '04)*. 318–329. https://doi.org/10.1145/964001.964028
[48] Peter W. Shor. 1994. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *Proceedings of the 34th annual Symposium on Foundations of Computer Science (SFCS '94)*. 124–134. https://doi.org/10.1109/SFCS.1994.365700
[49] 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 (Nov. 2020), 014003. https://doi.org/10.1088/2058-9565/ab8e92
[50] Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro. 2010. Quantum addition circuits and unbounded fan-out. *Quantum Information and Computation* 10, 9 (Sep. 2010), 872–890. https://doi.org/10.26421/QIC10.9-10-12
[51] Robert Endre Tarjan. 1981. Fast Algorithms for Solving Path Problems. *J. ACM* 28, 3 (Jul. 1981), 594–614. https://doi.org/10.1145/322261.322273
[52] Robert Endre Tarjan. 1981. A Unified Approach to Path Problems. *J. ACM* 28, 3 (Jul. 1981), 577–593. https://doi.org/10.1145/322261.322272
[53] John van de Wetering. 2024. Private communication.
[54] Vivien Vandaele. 2024. Lower T-count with faster algorithms. (2024). arXiv:2407.08695
[55] Renaud Vilmart. 2021. The Structure of Sum-Over-Paths, its Consequences, and Completeness for Clifford. In *Foundations of Software Science and Computation Structures (FoSSaCS '21)*. 531–550. https://doi.org/10.1007/978-3-030-71995-1_27
[56] Amanda Xu, Abtin Molavi, Lauren Pick, Swamit Tannu, and Aws Albarghouthi. 2023. Synthesizing Quantum-Circuit Optimizers. *Proceedings of the ACM on Programming Languages* 7, PLDI, Article 140 (Jun. 2023), 25 pages. https://doi.org/10.1145/3591254
[57] Mingsheng Ying. 2012. Floyd–Hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems* 33, 6, Article 19 (Jan. 2012), 49 pages. https://doi.org/10.1145/2049706.2049708
[58] Nengkun Yu and Jens Palsberg. 2021. Quantum abstract interpretation. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '21)*. 542–558. https://doi.org/10.1145/3453483.3454061
[59] Fang Zhang and Jianxin Chen. 2019. Optimizing T gates in Clifford+T circuit as $\pi/4$ rotations around Paulis. (2019). arXiv:1903.12456

A Additional experimental results

In this appendix we give the full results of our experimental evaluation. Table 3 reports the total gate counts. Queso+PP refers to Queso with an additional phase folding pre-processing step via Feynman. Queso was allowed to run for 2 hours and other tools were timed out after 2 hours. The vast majority of optimization runs from all other tools completed within seconds. The FastTODD algorithm includes a phase folding step which is equivalent to the t-count optimization in PyZX, as well as a gadgetization step which adds a large number (linear in the circuit length) of ancillas to the circuit. Only results computed by Feynman ($\mathbf{PF}_{\text{Aff}}$, $\mathbf{PF}_{\text{Quad}}$, and $\mathbf{PF}_{\text{Pol}}$) were formally verified for correctness.

Table 3. Complete results of our experimental evaluation.

Benchmark	n	Original		voqc		queso		queso+PP		FastTODD		$\mathbf{PF}_{\text{Aff}}$		$\mathbf{PF}_{\text{Quad}}$		$\mathbf{PF}_{\text{Pol}}$		$\mathbf{PF}_{\text{Pol}}$+FastTODD	
		# gates	# T	# gates	# T	# gates	# T	# gates	# T	# gates	# T	# gates	# T	# gates	# T	# gates	# T	# gates	# T
Grover_5	9	831	336	626	172	748	290	660	176	7313	143	696	148	696	148	586	**0**	10	**0**
Mod 5_4	5	63	28	54	16	45	10	33	10	54	**7**	52	8	52	8	52	8	61	**7**
VBE-Adder_3	10	150	70	97	24	93	28	76	24	387	**19**	117	24	52	24	117	24	365	**19**
CSLA-MUX_3	15	170	70	164	64	148	64	148	64	946	**39**	162	60	162	60	162	60	940	**39**
CSUM-MUX_9	30	420	196	308	84	406	168	252	84	1526	**71**	336	84	336	84	336	84	3322	**71**
QCLA-Com_7	24	443	203	332	99	335	135	272	95	1453	**59**	369	94	369	94	269	94	3161	61
QCLA-Mod_7	26	884	413	753	259	784	321	644	245	10532	**159**	770	237	770	237	770	237	19339	161
QCLA-Adder_10	36	521	238	449	162	466	198	394	164	2864	**109**	466	162	466	162	466	162	4626	**109**
Adder_8	24	900	399	748	223	729	265	596	219	8569	**119**	737	173	737	173	737	173	9401	121
RC-Adder_6	14	200	77	175	47	179	65	152	47	627	**37**	183	47	183	47	183	47	1185	**37**
Mod-Red_21	11	278	119	227	73	229	75	196	73	1183	**51**	245	73	245	73	245	73	1853	**51**
Mod-Mult_55	9	119	49	99	35	98	39	92	35	240	**17**	116	35	114	35	114	35	302	**17**
Mod-Adder_1024	28	4285	1995	3419	1035	3905	1746	3197	1255	TIMEOUT		3363	1011	3337	1005	3115	**923**	TIMEOUT	
GF(2^4)-Mult	12	225	112	192	68	198	82	176	68	667	**49**	194	68	194	68	194	68	632	**49**
GF(2^5)-Mult	15	347	175	292	115	299	135	273	115	1283	81	304	115	304	115	304	115	979	**75**
GF(2^6)-Mult	18	495	252	410	150	441	198	381	150	1607	113	414	150	414	150	414	150	2269	**111**
GF(2^7)-Mult	21	669	343	549	217	594	273	516	214	1745	155	553	217	553	217	553	217	2334	**141**
GF(2^8)-Mult	27	883	448	717	270	803	360	680	264	3861	205	709	264	709	264	709	264	3906	**203**
GF(2^9)-Mult	24	1095	567	887	351	943	459	835	351	4336	257	889	351	889	351	889	351	4021	**255**
GF(2^{10})-Mult	27	1347	700	1084	410	1213	568	1008	410	6418	315	1088	410	1088	410	1088	410	5043	**313**
GF(2^{16})-Mult	48	3435	1792	2715	1050	3295	1536	2613	1040	15641	**797**	2703	1040	2703	1040	2703	1040	19663	803
GF(2^{32})-Mult	96	13562	7168	10594	4152	13050	6144	10563	**4128**	TIMEOUT		10562	**4128**	10562	**4128**	10562	**4128**	TIMEOUT	
Ham_15 (low)	17	443	161	391	101	403	115	372	101	2679	**77**	391	97	391	97	391	97	2911	**77**
Ham_15 (med)	17	1272	574	946	266	1163	496	889	292	6315	**137**	901	212	901	212	895	210	8280	140
Ham_15 (high)	20	5308	2457	3832	1057	4768	2093	3988	1555	TIMEOUT		3969	1019	3914	**1007**	3927	**985**	TIMEOUT	
HWB_6	7	259	105	231	75	225	73	225	75	1328	**51**	237	75	237	75	237	75	1478	**51**
QFT_4	5	179	69	158	67	176	67	168	67	308	**53**	173	67	172	65	172	65	927	54
$\Lambda_3(X)$	5	45	21	42	15	40	15	35	15	90	**13**	42	15	42	15	42	15	140	**13**
$\Lambda_4(X)$	7	75	35	69	23	65	23	55	23	218	**19**	69	23	69	23	69	23	256	**19**
$\Lambda_5(X)$	9	105	49	96	31	90	31	75	31	358	**25**	96	31	96	31	96	31	415	**25**
$\Lambda_{10}(X)$	19	255	119	231	71	215	71	175	71	1211	**55**	231	71	231	71	231	71	1558	**55**
$\Lambda_3(X)$ (dirty)	5	60	28	52	16	48	16	38	16	112	**13**	52	16	52	16	52	16	110	**13**
$\Lambda_4(X)$ (dirty)	7	114	56	98	28	89	28	68	28	246	23	99	28	97	24	97	24	292	**20**
$\Lambda_5(X)$ (dirty)	9	170	84	144	40	130	60	98	40	442	33	146	40	142	32	142	32	528	**28**
$\Lambda_{10}(X)$ (dirty)	19	450	224	374	100	335	160	248	100	1517	83	381	100	367	72	367	72	2361	**68**
FP-renorm	10	266	112	254	94	264	108	251	94	2851	69	248	81	248	81	246	71	1425	**56**

Received 2024-07-11; accepted 2024-11-07