On the Principles of Differentiable Quantum Programming Languages

Shaopeng Zhu, Shih-Han Hung, Shouvanik Chakrabarti, Xiaodi Wu
University of Maryland, College Park, USA

Abstract
Variational Quantum Circuits (VQCs), or the so-called quantum neural-networks, are predicted to be one of the most important near-term quantum applications, not only because of their similar promises as classical neural-networks, but also because of their feasibility on near-term noisy intermediate-size quantum (NISQ) machines. The need for gradient information in the training procedure of VQC applications has stimulated the development of auto-differentiation techniques for quantum circuits. We propose the first formalization of this technique, not only in the context of quantum circuits but also for imperative quantum programs (e.g., with controls), inspired by the success of differentiable programming languages in classical machine learning. In particular, we overcome a few unique difficulties caused by exotic quantum features (such as quantum no-cloning) and provide a rigorous formulation of differentiation applied to bounded-loop imperative quantum programs, its code-transformation rules, as well as a sound logic to reason about their correctness. Moreover, we have implemented our code transformation in OCaml and demonstrated the resource-efficiency of our scheme both analytically and empirically. We also conduct a case study of training a VQC instance with controls, which shows the advantage of our scheme over existing auto-differentiation for quantum circuits without controls.

CCS Concepts: • Theory of computation → Quantum information theory; • Denotational semantics; • Computing methodologies → Machine learning.

Keywords: quantum programming languages, differentiable programming languages, quantum machine learning

ACM Reference Format:
Shaopeng Zhu, Shih-Han Hung, Shouvanik Chakrabarti, Xiaodi Wu. 2020. On the Principles of Differentiable Quantum Programming Languages. In Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI ’20), June 15–20, 2020, London, UK. ACM, New York, NY, USA, 25 pages. https://doi.org/10.1145/3385412.3386011

1 Introduction

Background. Recent years have witnessed the rapid development of quantum computing, with practical advances coming from both research and industry. Quantum programming is one topic that has been actively investigated. Early work on language design [24, 35, 42, 43, 47] has been followed up recently by several implementations of these languages, including Quipper [26], Scaffold [2], LIQUi [52], Q# [49], and QWIRE [36]. Extensions of program logics have also been proposed for verification of quantum programs [4, 10, 11, 19, 29, 30, 55, 57]. See also surveys [20, 46, 56].

With the availability of prototypes of quantum machines, especially the recent establishment of quantum supremacy [3], the research of quantum computing has entered a new stage where near-term Noisy Intermediate-Scale Quantum (NISQ) computers [40], e.g., the 53-qubit quantum machines from Google [3] and IBM [22], become the important platform for demonstrating quantum applications. Variational quantum circuits (VQCs) [17, 18, 38], or the so-called quantum neural networks, are predicted to be one of the most important applications on NISQ machines. It is not only because VQCs bear a lot of similar promises like classical neural networks as well as potential quantum speed-ups from the perspective of machine learning (e.g., see the survey [9]), but also because VQC is, if not the only, one of the few candidates that can be implemented on NISQ machines. Because of this, a lot of study has already been devoted to the design, analysis, and small-scale implementation of VQCs (e.g., see the survey [7]).

Typical VQC applications replace classical neural networks, which are just parameterized classical circuits, by quantum circuits with classically parameterized unitary gates. Namely, one will have a "quantum" mapping from input to
output replacing classical mapping in machine learning applications. An important component of these applications is a training procedure which optimizes a *loss function* that now depends on the *read-outs* and the *parameters* of VQCs.

Gradient-based approaches are widely used in the training procedure. However, computing these gradients of loss functions from quantum circuits has a similar complexity of simulating quantum circuits, which is infeasible for classical computation. Thus, the ability of evaluating these “quantum” gradients efficiently by quantum computation is critical for the scalability of VQC applications.

Fortunately, analytical formulas of gradients used in VQCs have been studied by [18, 21, 32, 44, 45]. In particular, Schuld et al. [44] proposed the so-called *phase-shift rule* that uses two quantum circuits to compute the partial derivative respective to one parameter for quantum circuits. One of the very successful tools for quantum machine learning, called PennyLane [8], implemented the phase-shift rule to achieve *auto differentiation* (AD) for the read-outs of quantum circuits. It also integrated automatic differentiation from established machine learning libraries such as TensorFlow or PyTorch for any additional classical calculation in the training procedure. However, none of these studies was conducted from the perspective of programming languages and no rigorous foundation or principles have been formalized.

**Motivations.** An important motivation of this paper is to provide a *rigorous formalization* of the auto-differentiation technique applied to quantum circuits. In particular, we will provide a formal formulation of quantum programs, their semantics, and the meaning of differentiation of them. We will also study the code-transformation rules for auto-differentiation and prove their correctness.

As we will highlight below, research on the formalization will encounter many new challenges that have not been considered or addressed by existing results [18, 21, 32, 44, 45]. Consider one of the basic requirements, e.g., *compositionality*. As we will show, differentiating the composition of quantum programs will necessarily involve running multiple quantum programs on copies of initial quantum states. How to represent the collection of quantum programs succinctly and also bound the number of required copies is a totally new question. Among our techniques to address this question, we also need to change the previously proposed construct, e.g., the phase-shift rule [44], to something different.

Moreover, we want to go beyond the restriction of quantum circuits. Our inspiration comes from classical machine learning examples that demonstrate the advantage of neural-networks with program features (e.g. controls) over the plain ones (e.g., classical circuits), e.g. [25, 27], which is also the major motivation of promoting the the paradigm shift from deep learning toward *differentiable programming*.

Augmenting VQCs with controls, at least for simple ones, is not only feasible on NISQ machines, but also a logical step for the study of their applications in machine learning. Therefore, we are inspired to investigate the *principles of differentiable quantum imperative languages* beyond circuits. Indeed, we conduct one such case study in Section 8.

**Research challenges & Solutions.** We will rely on a few notations that should be self-explanatory. Please refer to a detailed preliminary on quantum information in Section 2. Let us start with a simple classical program

$$\text{MUL} \equiv v_3 = v_1 \times v_2,$$

where $v_3$ is the product of $v_1$ and $v_2$. Consider the differentiation with respect to $\theta$, we have

$$\frac{\partial}{\partial \theta} (\text{MUL}) \equiv v_3 = v_1 \times v_2;$$

$$\dot{v}_3 = \dot{v}_1 \times v_2 + v_1 \times \dot{v}_2,$$

where MUL keeps track of variables $v_1, v_2, v_3$ and their derivatives $\dot{v}_1, \dot{v}_2, \dot{v}_3$ at the same time. One simple yet important observation is that classical variables $v_1, v_2, v_3$ are real-valued and can be naturally differentiated.

Given that quantum states are represented by matrices, what are the natural quantities to differentiate in the quantum setting? One natural choice from the principles of quantum mechanics is the (classical) read-outs of quantum systems through measurements, which we formulate as the *observable semantics* of quantum programs. This natural choice also serves the purpose of gradient computation of loss functions in quantum machine learning, which are typically defined in terms of these read-outs. We directly model the parameterization of quantum programs after VQCs, i.e., each unitary gate becomes *classically parameterized*.¹

To model the meaning of one quantum program computing the derivative of another, we define the *differential semantics* of programs. There is a subtle quantum-unique design choice. The observable semantics of any quantum program will depend on the observable and its input state. Thus, any program computing its derivative could potentially depend on these two extra factors. We find out this potential dependence is undesirable and propose the strongest possible definition: i.e., one derivative computing program should work for *any pair* of observables and input states. We demonstrate that this strong requirement is not only achievable but also critical for the composition of auto differentiation.

We are ready to describe the technical challenges for the compositionality. Consider the following quantum program:

$$\text{QMUL} \equiv U_1(\theta); U_2(\theta),$$

which performs $U_1(\theta)$ and $U_2(\theta)$ gates sequentially. Note that gate application is matrix multiplication in the quantum setting. Roughly speaking, if the product rule of differentiation

¹The above modeling of quantum programs is very different from classical ones. It is unclear whether any reasonable analogue of classical chain-rule and forward/backward mode can exist within quantum programs.
(as exhibited in (1.3)) remains in the quantum setting, at least symbolically, then one should expect \( \frac{\partial}{\partial \theta} (\text{QMUL}) \) contains

\[
\frac{\partial}{\partial \theta} (U_1(\theta); U_2(\theta)) \text{ and } U_1(\theta); \frac{\partial}{\partial \theta} (U_2(\theta))
\]  

(1.5)
two different parts as sub-programs similarly in (1.3).

However, we cannot run \( \frac{\partial}{\partial \theta} (U_1(\theta); U_2(\theta); \frac{\partial}{\partial \theta} (U_2(\theta)) \) together due to the quantum no-cloning theorem [54]. This is simply because they share the same initial state and we cannot clone two copies of it. Note that this is not an issue classically as we can store all \( v_i, \tilde{v}_i \) at the same time as in (1.3). As a result, quantum differentiation needs to run multiple (sub-)programs on multiple copies of the initial state.

This change poses a unique challenge for differentiation of quantum composition: (1) we hope to have a simple scheme of code transformation, ideally close-to-classical, for intuition and easy implementation of the compiler, whereas it needs to express correctly the collection of quantum programs during code transformation; (2) for the purpose of efficiency, we also want to reasonably bound the number of required copies of the initial states, which roughly refers to the number of different quantum programs in the collection.

We develop a few techniques to achieve both goals at the same time. First, we propose the so-called additive quantum programs as a succinct intermediate representation for the collection of programs during the code transformation. Now the entire differentiation procedure will be divided into two steps: (1) all code transformations happen on additive programs and are very similar to classical ones (see Figure 4); (2) the collection of programs can be recovered by a compilation procedure from any additive program. Additive quantum programs are equipped with a new sum operation that models the multiple choices as exhibited in (1.5), which resembles a similar idea in the differential lambda-calculus [14].

Second, we also design a new rule for \( \frac{\partial}{\partial \theta} (U(\theta)) \) which is slightly different from [18, 21, 32, 44, 45]. The existing phase-shift rule makes use of two quantum circuits for one differentiation, which causes a lot of inconvenience in the formulation and potential trouble for efficiency. Instead, we use only one extra ancilla as the control qubit to create a superposition of two quantum circuits and effectively achieve the same differentiation with only one quantum circuit. We also conduct a careful resource analysis of our differentiation procedure and show the number of required copies of initial states is reasonable comparing to the classical setting. The correctness of the code transformation of composition critically relies on our design choice as well as the strong definition related to the differential semantics.

With the previous setup, we can naturally build the differentiation for quantum controls (i.e., the condition statement). Note that a general solution for classical controls is unknown [6] due to the non-smoothness of the guard. Similar to the classical setting [39], we only provide a solution to deal with bounded loops and leave it open for general ones.

**Contributions.** We formulate the parameterized quantum bounded while-programs with classically parameterized unitary gates modelled after VQCs [17, 33, 38] and their realistic examples on ion-trap machines, e.g. [58], in Section 3.

In Section 4, we illustrate our design of additive quantum programs. Specifically, we add the syntax \( P_1 + P_2 \) to represent the either-or choice between \( P_1 \) and \( P_2 \) in (1.5). We formulate its semantics and compilation rules that map additive programs into collections of normal ones for our purpose.

In Section 5, we formulate the observable and the differential semantics of quantum programs and formally define the meaning of program \( S'(\theta) \) computing the differential semantics of \( S(\theta) \) in the strongest possible sense.

In Section 6, we show that such a strong requirement is indeed achievable by demonstrating the code-transformation rules for the differentiation procedure. Thanks to the use of additive quantum programs, the code transformation is much simplified and as intuitive as classical ones. We develop a logic with the judgement \( S'(\theta)|S(\theta) \) stating that \( S'(\theta) \) computes the differential semantics of \( S(\theta) \). We prove it sound and use it to show the correctness of the code transformation.

In Section 7, we conduct a resource analysis to further justify our design. We show that the occurrence count of parameters capture the extra resource required in both the classical auto-differentiation and our scheme. Hence, our resource cost is reasonable compared with the classical setting.

Finally, in Section 8, we demonstrate the implementation of our code transformation in OCaml and apply it to the training of one VQC instance with controls via classical simulation. Specifically, this instance shows an advantage of controls in machine learning tasks, which implies the advantage of our scheme over previous ones that cannot handle controls. We have also empirically verified the resource-efficiency of our scheme on representative VQC instances.

**Related classical work.** There is an extensive study of automatic differentiation (AD) or differentiable programming in the classical setting (e.g., see books [12, 28]). The most relevant to us are those studies from the programming language perspective. AD has traditionally been applied to imperative programs in both the forward mode, e.g. [31, 53], and the reverse mode, e.g., [48]. The famous backpropagation algorithm [41] is also a special case of reserve-mode AD used to compute the gradient of a multi-layer perceptron. AD has also been recently applied to functional programs [15, 16, 37]. Motivated by the success of deep learning, there is significant recent interest to develop both the theory and the implementation of AD techniques. Please refer to the survey [5] and the keynote talk at POPL’18 [39] and [1] for more details.

## 2 Quantum Preliminaries

We present basic quantum preliminaries here (a summary of notation in Table 1). Details are deferred to Appendix A.
2.1 Math Preliminaries

Let $n$ be a natural number. We refer to the complex vector space $\mathbb{C}^n$ as an $n$-dimensional Hilbert space $\mathcal{H}$. We use $|\psi\rangle$ to denote a complex vector in $\mathbb{C}^n$. The Hermitian conjugate of $|\psi\rangle \in \mathbb{C}^n$ is denoted by $\langle \psi |$. The inner product of $|\psi\rangle$ and $|\phi\rangle$, defined as the product of $|\phi\rangle$ and $|\psi\rangle$, is denoted by $\langle \psi | \phi \rangle$. The norm of a vector $|\psi\rangle$ is denoted by $|||\psi||| = \sqrt{\langle \psi | \psi \rangle}$.

We define operators as linear maps between Hilbert spaces, which can be represented by matrices for finite dimensions. Let $A$ be an operator and its Hermitian conjugate $A^\dagger$. $A$ is Hermitian if $A = A^\dagger$. The trace of $A$ is the sum of the entries on the main diagonal, i.e., $\text{tr}(A) = \sum_i A_{ii}$. $\langle \psi | A | \phi \rangle$ denotes the inner product of $|\psi\rangle$ and $A|\phi\rangle$. Hermitian operator $A$ is positive semidefinite if for all vectors $|\psi\rangle \in \mathcal{H}$, $\langle \psi | A | \psi \rangle \geq 0$.

2.2 Quantum States and Operations

The state space of a qubit is a 2-dimensional Hilbert space. Two important orthonormal bases of a qubit system are: the computational basis with $|0\rangle = (1, 0)^\dagger$ and $|1\rangle = (0, 1)^\dagger$; the ± basis, consisting of $|\pm\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$ and $|\mp\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$.

A pure quantum state is a unit vector $|\psi\rangle$. A mixed state, which refers to an ensemble of pure states $\{|p_i\rangle, |\psi_i\rangle\}_i$ (each with probability $p_i$), can be represented by a density operator that is a trace-one positive semidefinite operator $\rho = \sum_i p_i |\psi_i\rangle \langle \psi_i |$. $\rho$ is a partial density operator if $\text{tr}(\rho) \leq 1$. The set of partial density operators on $\mathcal{H}$ is denoted by $\mathcal{D}(\mathcal{H})$.

Operations on quantum systems can be characterized by unitary operators. Denoting the set of linear operators on $\mathcal{H}$ as $L(\mathcal{H})$, an operator $U \in L(\mathcal{H})$ is unitary if $U^\dagger U = U U^\dagger = I_H$. A unitary evolves a pure state $|\psi\rangle$ to $U|\psi\rangle$, or a density operator $\rho$ to $U \rho U^\dagger$. Common unitary operators include the Hadamard operator $H$, which transforms between the computational and the ± basis via $H|0\rangle = |+\rangle$ and $H|1\rangle = |-\rangle$; the Pauli $X$ operator which performs a bit flip, i.e., $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$; Pauli $Z$ which performs a phase flip, i.e., $Z|0\rangle = |0\rangle$ and $Z|1\rangle = |1\rangle$; CNOT gate mapping $|00\rangle \mapsto |00\rangle$, $|01\rangle \mapsto |01\rangle$, $|10\rangle \mapsto |11\rangle$, $|11\rangle \mapsto |10\rangle$. More generally, evolution of a quantum system can be characterized by an admissible superoperator $\mathcal{E}$, namely a completely-positive and trace-non-increasing linear map from $\mathcal{D}(\mathcal{H})$ to $\mathcal{D}(\mathcal{H})$.

For every superoperator $\mathcal{E}$, there exists a set of Kraus operators $\{E_k\}_k$ such that $\mathcal{E}(\rho) = \sum_k E_k \rho E_k^\dagger$ for any input $\rho \in \mathcal{D}(\mathcal{H})$. The Kraus form of $\mathcal{E}$ is therefore $\mathcal{E} = \sum_k E_k \circ E_k^\dagger$. The Schrödinger-Heisenberg dual of a superoperator $\mathcal{E} = \sum_k E_k \circ E_k^\dagger$, denoted by $\mathcal{E}^\star$, is defined as follows: for every state $\rho \in \mathcal{D}(\mathcal{H})$ and any operator $A$, $\text{tr}(A \mathcal{E}(\rho)) = \text{tr}(\mathcal{E}^\star(A) \rho)$. The Kraus form of $\mathcal{E}^\star$ is $\sum_k E_k^\dagger \circ E_k$.

2.3 Quantum Measurements

Quantum measurements extracts classical information out of quantum systems. A quantum measurement on a system over Hilbert space $\mathcal{H}$ can be described by a set of linear operators $\{M_m\}_m$ with $\sum_m M_m^\dagger M_m = I_H$ (identity matrix on $\mathcal{H}$). If we perform a measurement $\{M_m\}_m$ on a state $\rho$, the outcome $m$ is observed with probability $p_m = \text{tr}(M_m \rho M_m^\dagger)$ for each $m$, and the post-measurement state collapses to $M_m \rho M_m^\dagger / p_m$.

3 Parameterized Quantum Bounded While-Programs

We adopt the bounded-loop variant of the quantum while-language developed by Ying [56], and augment it by parameterizing the unitaries, as this provides sufficient expressibility for parameterized quantum operations: indeed, abort, skip and initialization behave independently of parameters, while “parameterized measurements” can be implemented with a regular measurement followed by a parameterized unitary.

From here onward, $\mathcal{V}$ is a finite set of variables, and $\theta$ a length-$k$ vector of real-valued parameters.

3.1 Syntax

Define $\text{Var}$ as the set of quantum variables. We use the symbol $q$ as a metavariable ranging over quantum variables and define a quantum register $\overline{q}$ to be a finite set of distinct variables. For each $q \in \text{Var}$, its state space is denoted by $\mathcal{H}_q$. The quantum register $\overline{q}$ is associated with the Hilbert space
A T-bounded, k-parameterized quantum \textit{while}-program is generated by the following syntax:

\begin{align*}
P(\theta) &:= \text{abort}(\phi) \mid \text{skip}(\phi) \mid q := 0 \mid \bar{\phi} := U(\theta)(\phi) \mid \text{case } M[\phi] = m \rightarrow P_m(\theta) \text{ end } \mid \\
& \quad \text{while}^{(T)} M[\phi] = 1 \text{ do } P_1(\theta) \text{ done,}
\end{align*}

\text{where}

\begin{align*}
\text{while}^{(1)} M[\phi] = 1 & \text{ do } P_1(\theta) \text{ done} \\
& \equiv \text{ case } M[\phi] = \{0 \rightarrow \text{skip, } 1 \rightarrow P_1(\theta); \text{abort}\}.
\end{align*}

\text{Unparameterized programs can be obtained by fixing } \theta^* \in \mathbb{R}^k \text{ in some } P(\theta). \text{ We denote the set of variables accessible to } P(\theta) \text{ as } \text{qVar}(P(\theta)); \text{ the collection of all } “T\text{-bounded while}$$\text{-}$$\text{programs } P(\theta) \text{ s.t. } \text{qVar}(P(\theta)) = \mathbb{T}^\ast = \text{q}-\text{while}^{(T)}(\theta), \text{ and similarly, the unparameterized one as } \text{q}-\text{while}^{(T)}.$$

Now let us formally define parameterization of unitaries: let \( \theta := (\theta_1, \ldots, \theta_k) \ (k \geq 1). \) A \textit{k-parameterized unitary} 
\( U(\theta) \) is a function \( \mathbb{R}^k \rightarrow \mathcal{U}(\mathcal{H}) \text{ s.t. } \) (1) given any \( \theta^* \in \mathbb{R}^k, \) 
\( U(\theta^*) \) is an unitary on \( \mathcal{H}, \) and (2) the parameterized-matrix representation of \( U(\theta) \) is entry-wise smooth.

An important family is the single-qubit rotations about the Pauli axis \( X, Y, Z \) with angle \( \theta \) (matrix exponential here):

\[
R_{\theta}(\sigma) := \exp \left( -\frac{i \theta}{2} \sigma \right), \quad \sigma \in \{X, Y, Z\}.
\]

One can also extend Pauli rotations to multiple qubits. For example, consider \textit{two-qubit coupling gates} \( \{R_{\theta;\sigma} := \exp(-i \theta \sigma \otimes \sigma)\}_{\sigma \in \{X, Y, Z\}}, \) which generate entanglement between two qubits. Combined with single-qubit rotations, they form a universal gate set for quantum computation. Another important feature of these gates is that they can already be reliably implemented in such as ion-trap quantum computers [58].

As a result, we will work mostly with these gates in the rest of this paper. However, note that one can easily add and study other parameterized gates in our framework as well.

The language constructed above is similar to their classical counterparts. \( 0 \) \texttt{abort} terminates the program, outputting \( 0 \in \mathcal{D}(\mathcal{H}_q). \) (1) \texttt{skip} does nothing to states in \( \mathcal{D}(\mathcal{H}_q). \) (2) \( q := 0 \) sets quantum variable \( q \) to the basis state \( 0. \) The underlying quantum procedure is to apply super-operators \( E_{q=0}^{\text{bool}}(\cdot) \) (or \( E_{q=0}^{\text{int}}(\cdot) \)) to \( q \) and identity operations to the rest of variables. The correlation between \( q \) and the rest of quantum variables could be potentially disturbed. (3) for any \( \theta^* \in \mathbb{R}^k, \) \( \bar{\phi} := U(\theta^*)(\phi) \) applies the unitary \( U(\theta^*) \) to the qubits in \( \bar{\phi}. \) (4) Sequencing has the same behavior as its classical counterpart. (5) for \( \theta^* \in \mathbb{R}^k, \) case \( M[\phi] = m \rightarrow P_m(\theta^*) \) end performs the measurement \( M = \{M_m\} \) on the qubits in \( \bar{\phi}, \) and executes program \( P_m(\theta^*) \) if the outcome of the measurement is \( m. \) The bar over \( m \rightarrow P_m \) indicates that there may be one or more repetitions of this expression. (6) \text{while}^{(T)} M[\phi] = 1 \text{ do } P_1(\theta^*) \text{ done} performs the measurement \( M = \{M_0, M_1\} \) on \( \bar{\phi}, \) and terminates if the outcome corresponds to \( M_0, \) or executes \( P_1(\theta^*) \) then reiterates \( (T \geq 2) \) \text{ aborts } \( (T = 1) \) otherwise. The program iterates at most \( T \) times.

We highlight two differences between quantum and classical while languages: (1) Qubits may only be initialized to the state \( |0\rangle. \) There is no quantum analogue for initialization to any expression (i.e. \( x := e \)) due to the non-cloning theorem of quantum states. Any state \( |\psi\rangle \in \mathcal{H}_q \) however, can be constructed by applying some unitary \( U \) to \( |0\rangle. \) (2) Evaluating the guard of a case statement or loop, which performs a measurement, potentially disturbs the state of the system.

### 3.2 Operational and Denotational Semantics

We present the \textit{operational semantics} of parameterized programs in Figure 1a. Transition rules are represented as \( (P, \rho) \rightarrow (P', \rho'), \) where \( (P, \rho) \) and \( (P', \rho') \) are quantum configurations. In configurations, \( P \) (or \( P' \)) could be a quantum program or the empty program \( \bot, \) and \( \rho \) and \( \rho' \) are partial density operators representing the current state. Intuitively, in one step, we can evaluate program \( P \) on input state \( \rho \) to program \( P' \) (or \( \bot \)) and output state \( \rho'. \) In order to present the rules in a non-probabilistic manner, the probabilities associated with each transition are encoded in the output partial density operator. For each index \( m \) of branches in a loop/control statement, the superoperator \( E_m \) is defined by \( E_m(\rho) = M_m \rho M_m^\dagger, \) yielding the post-measurement state.

We present the \textit{denotational semantics} of parameterized programs in 1b, defining \( \llbracket P \rrbracket \) as a superoperator on \( \rho \in \mathcal{H}_q \) [56]. For more details we refer the reader to Ying [55, 56].

We have the following connection between the denotational semantics and operational for parameterized programs: in short, the meaning of running program \( P(\theta^*) \) on input state \( \rho \) and any \( \theta^* \in \mathbb{R}^k \) is the sum of all possible output states with multiplicity, weighted by their probabilities.

**Proposition 3.1** ([56]). \( \forall P(\theta) \in \text{q}-\text{while}^{(T)}(\theta), \) and any specific \( \theta^* \in \mathbb{R}^k, \rho \in \mathcal{D}(\mathcal{H}_q), \)

\[
\llbracket P(\theta^*) \rrbracket(\rho) = \sum \{ |\rho' : (P(\theta^*), \rho) \rightarrow^* (\bot, \rho')| \}.
\]

Here \( \rightarrow^* \) is the reflexive, transitive closure of \( \rightarrow \) and \( \{ | \cdot | \} \) denotes a multi-set.

We close the section with a notion arising from the following observation: some programs, while syntactically not \texttt{abort}(\bar{\phi}), semantically aborts. Simple examples include \( U(\theta); \texttt{abort} \) or a case sentence that has \texttt{abort} on each branch. These

---

\(^{2}\) If type \( q = \text{bool} \) then \( \mathcal{H}_q = \text{span} \{ |0\rangle, |1\rangle \}. \) If type \( q = \text{bool} \) then \( \mathcal{H}_q = \text{span} \{ |n\rangle : n \in \{-N, \ldots, N\} \} (N \in \mathbb{Z}^\ast) \) for some finite \( N. \) We require the Hilbert space to be finite dimensional for implementation.

\(^{3}\) \( E_{q=0}^{\text{int}}(\rho) = |0\rangle\langle 0| \rho |0\rangle\langle 0| + |0\rangle\langle 1| \rho |1\rangle\langle 0| + |1\rangle\langle 0| \rho |0\rangle\langle 0| (N \in \mathbb{Z}^\ast). \)

\(^{4}\) Recall that, fixing arbitrary \( \theta^* \in \mathbb{R}^k, \) both semantics reduce to those of unparameterized programs, so for compactness we write \( P \) for \( P(\theta^*) \), etc.
(Abort) \(\text{abort}[\bar{q}], \rho) \rightarrow (\downarrow, 0)\)

(Skip) \(\text{skip}[\bar{q}], \rho) \rightarrow (\downarrow, \rho)\)

(Init) \(q := [0], \rho) \rightarrow (\downarrow, \rho^0)\)

where \(\rho^0 q = \begin{cases} \mathcal{E}_q^{\text{bool}}(\rho) & \text{if type}(q) = \text{Bool} \\ \mathcal{E}_q^{\text{Bdd Int}}(\rho) & \text{if type}(q) = \text{Bdd Int} \end{cases}\)

(Unitary) \(\bar{q} := U(\theta)[\bar{q}], \rho) \rightarrow (\downarrow, U(\theta)P^{\uparrow}(\theta))\)

\(P_1(\theta), \rho) \rightarrow (P_1(\theta), \rho)\)

(Sequence) \((P_1(\theta); P_2(\theta)), \rho) \rightarrow (P_1(\theta); P_2(\theta), \rho)\)

(Case m) \(\text{case } M[\bar{q}] = m \rightarrow P_m(\theta) \text{ end }, \rho) \rightarrow \langle P_m(\theta).E_m(\rho) \rangle, \forall \text{ outcome } m \text{ of } M = \{M_m\}\)

(While\(^T\)) 0 \((\text{while}^T) M[\bar{q}] = 1 \text{ do } P_1(\theta) \text{ done }, \rho) \rightarrow (\downarrow, E_0(\rho))\)

(While\(^T\)) 1 \((\text{while}^T) M[\bar{q}] = 1 \text{ do } P_1(\theta) \text{ done }, \rho) \rightarrow \langle P_1(\theta); \text{while}^{T-1}(\theta), E_1(\rho) \rangle\)

(a) \(\llbracket \text{abort}[\bar{q}] \rrbracket_\rho = 0\)

\(\llbracket \text{skip}[\bar{q}] \rrbracket_\rho = \rho\)

\(\llbracket q := [0] \rrbracket_\rho = \mathcal{E}_q^{\text{bool}}(\rho) \text{ or } \mathcal{E}_q^{\text{Bdd Int}}(\rho)\)

\(\llbracket \bar{q} := U(\theta)[\bar{q}] \rrbracket_\rho = U(\theta)P^{\uparrow}(\theta)\)

\(\llbracket P_1(\theta); P_2(\theta) \rrbracket_\rho = \langle P_2(\theta) \rangle \llbracket P_1(\theta) \rrbracket_\rho\)

\(\llbracket \text{case } M[\bar{q}] = m \rightarrow P_m(\theta) \text{ end } \rrbracket_\rho = \mathcal{E}_m(\rho)\)

\(\llbracket \text{while}^T(\theta) M[\bar{q}] = 1 \text{ do } P_1(\theta) \text{ done } \rrbracket_\rho = \langle \sum_{n=0}^{T-1} E_n \circ \mathcal{E}_1)^n(\rho) \rangle\)

(b) \(\mathcal{E}_q^{\text{bool}}(\rho) = \begin{cases} \mathcal{E}_q^{\text{Bdd Int}}(\rho) & \text{if type}(q) = \text{Bdd Int} \\ \mathcal{E}_q^{\text{bool}}(\rho) & \text{if type}(q) = \text{Bool} \end{cases}\)

Figure 1. Parameterized \(T\)-bounded quantum while programs: (a) operational semantics (b) denotational semantics.

Figure 2. additive parameterized quantum bounded while-program: operational semantics. We fix \(\theta \in \mathbb{R}^k\) and inherit all the other rules from parameterized programs in Fig. 1a.

4 Additive Parameterized Quantum Bounded While-Programs

We introduce a variant of additive quantum programs as a succinct way to describe the collection of programs that are necessary to compute the derivatives. To that end, we introduce our design of the syntax and the semantics of additive quantum programs as well as a compilation method that turns any additive quantum program into a collection of normal programs for the actual computation of derivatives.

4.1 Syntax

We adopt the convention to use underlines to indicate additive programs, such as \(P(\theta)\), to distinguish from normal program \(P(\theta)\). The syntax of \(P(\theta)\) is given by

\[\begin{align*}
P(\theta) &::= \text{abort}[\bar{q}] \mid \text{skip}[\bar{q}] \mid q := [0] \mid \bar{q} := U(\theta)[\bar{q}] \mid P_1(\theta); P_2(\theta) \mid \text{case } M[\bar{q}] = m \rightarrow P_m(\theta) \text{ end } \\
& \text{ while}^T(\theta) M[\bar{q}] = 1 \text{ do } P_1(\theta) \text{ done } \mid P_1(\theta) + P_2(\theta),
\end{align*}\]

where the only new syntax + is the additive choice. Intuitively, \(P_1(\theta) + P_2(\theta)\) allows the program to either execute \(P_1(\theta)\) or \(P_2(\theta)\) nondeterministically. The denotational semantics will include all possible execution traces. We assume + has lower precedence order than composition, and is left associative.\(^3\) If \(P(\theta) = P_1(\theta) + P_2(\theta)\), then qVar\(P(\theta)\) \(\equiv\) \(q\text{Var}(P_1(\theta)) \cup q\text{Var}(P_2(\theta))\). Denote the collection of all nondeterministic \(P(\theta)\) s.t. \(q\text{Var}(P(\theta)) = \emptyset\) as add-\(q\text{-while}^T(\theta)\).

4.2 Operational and Denotational Semantics

We exhibit operational semantics in Figure 2 and define a similar denotational semantics for any \(P(\theta) \in \text{add-}\text{q-while}^T(\theta)\).

Definition 4.1 (Denotational Semantics). Fix \(\theta\). \(\forall \rho \in \mathcal{D}(\mathcal{H}_\pi)\),

\[\llbracket P(\theta) \rrbracket_\rho(\rho) = \{ |\rho' : \langle P(\theta), \rho \rangle \rightarrow^* (\downarrow, \rho') | \}.\]

Note that there is no sum in (4.1) compared with (3.3). This is because we want to capture the behavior of + by storing all possible execution traces in a multi-set. This resembles the idea of the sum operator in the differential lambda-calculus [14].

\(^3\)E.g., \(X + Y + Z = X + (Y + Z)\), \(X + Y + Z = (X + Y) + Z\).
4.3 Compilation Rules

We exhibit the compilation rules in Figure 3 as a way to transform an additive program \( P(\theta) \) into a multiset of normal programs. The compiled set of programs will be later used in the actual implementation of the differentiation procedure. Our compilation rule is also well-defined as it is compatible with the denotational semantics and operational semantics of \( P(\theta) \) in the following sense:

**Proposition 4.2.** Denoting with \( \bigcup \) the union of multisets, then for any \( \rho \in \mathcal{D}(\mathcal{H}_T) \),
\[
\{ |\rho^{'}, \rho^{'} \neq 0, \rho^{'} \in \mathcal{D}(P(\theta)) | \rho \} = \bigcup_{\mathcal{Q}(\theta) \in \text{Compile}(P(\theta))} \{ |\rho^{'}, \rho^{'} \neq 0, \rho^{'} \in \mathcal{D}(P(\theta)) | \rho \}.
\]

**(4.2)**

*Proof:* Structural Induction. See Appendix C.1 for details. \( \square \)

Note that (4.2) removes 0 from the multi-set as we are only interested in non-trivial final states. Moreover, in Compile\((P(\theta))\), some programs may essentially abort (Definition 3.2). For implementation, we are interested in the number of \( Q(\theta) \in \text{Compile}(P(\theta)) \) that do not essentially abort:

**Definition 4.3.** The number of non-aborting programs of \( P(\theta) \), denoted as \( |\mathcal{P}(\theta)| \), is defined as
\[
|\mathcal{P}(\theta)| = |\text{Compile}(P(\theta))| - | \{ Q(\theta) \in \text{Compile}(P(\theta)) : Q(\theta) \text{ essentially aborts} \} |.
\]

where \( |C| \) is the cardinality of a multiset \( C \) and \( C_0 \backslash C_1 \) denotes the multiset difference of \( C_0 \) and \( C_1 \).

We remark that \( |\mathcal{P}(\theta)| \) could be exponentially large for general \( P(\theta) \), e.g., \( P(\theta) \equiv (Q_1 + R_1); \ldots; (Q_n + R_n) \). However, as we show in Section 7, for instances of additive programs from differentiation, this number is well bounded. (i.e., instances with exponential blow-up are irrelevant in our context.)

**Example 4.1 (Generic-Case).** Consider the following simple program with the case statement
\[
P(\theta) = \text{case } M[\tilde{3}] = 0 \rightarrow P_1(\theta) + P_2(\theta),
\]
where \( P_1(\theta), P_2(\theta), P_3(\theta) \in q\text{-}while^T(\theta) \), none of them essentially aborts, and each of \( P_1(\theta), P_2(\theta), P_3(\theta) \) contains no control gates. Then for any \( \rho \in \mathcal{D}(\mathcal{H}_T) \), fixing \( \theta \) we have
\[
\langle P(\theta^*) \rangle, \rho \xrightarrow{\text{(Case m)}} \langle P_1(\theta^*) + P_2(\theta^*), M_0 \rho M_0^* \rangle
\]
\[
\xrightarrow{\text{(Sum)}} \langle P_1(\theta^*), M_0 \rho M_0^* \rangle
\]
\[
\xrightarrow{\text{(*)}} \langle \bot, [\mathcal{P}(\theta^*)]([M_0 \rho M_0^*]) \rangle;
\]
\[
\langle P_2(\theta^*), M_0 \rho M_0^* \rangle
\]
\[
\xrightarrow{\text{(Case m)}} \langle P_3(\theta^*), M_1 \rho M_1^* \rangle \xrightarrow{\text{(*)}} \langle \bot, [\mathcal{P}(\theta^*)]([M_1 \rho M_1^*]) \rangle;
\]
\[
\langle P_3(\theta^*), M_1 \rho M_1^* \rangle
\]
\[
\xrightarrow{\text{(*)}} \langle \bot, [\mathcal{P}(\theta^*)]([M_1 \rho M_1^*]) \rangle
\]

**Hence by Definition 4.1.**
\[
\mathcal{P}(\theta^*) = \{ |\mathcal{P}(\theta^*)| ([M_0 \rho M_0^*]), \mathcal{P}(\theta^*) ([M_0 \rho M_0^*]), \mathcal{P}(\theta^*) ([M_1 \rho M_1^*]) \}
\]

We verify computation results from the compilation rules are consistent with this. Writing “compilation rule” as “CP” for short, one observes \( \text{Compile}(P_1(\theta) + P_2(\theta)) \xrightarrow{\text{CP:Sum}} \{ |P_1(\theta), P_2(\theta)| \} \), while \( \text{Compile}(P_3(\theta)) = \{ |P_3(\theta)| \} \) since we assumed non-essentially-abortness. Apply our “fill and break” procedure to obtain \( C_0 = \{ |P_1(\theta), P_2(\theta)| \} \), \( C_1 = \{ |P_3(\theta), \text{abort}^T(\theta)| \} \).
We define the observable semantics of both normal (denoted by \(P(\theta), P'(\theta)\)) and additive (denoted by \(S(\theta), S'(\theta)\)) parameterized programs as follows.

**Definition 5.1 (Observable Semantics).** \(\forall P(\theta) \in q\text{-}while^{T}_\uparrow(\theta),\) any observable \(O \in O_\uparrow\) and input state \(\rho \in D(H_\uparrow),\) the observable semantics of \(P,\) denoted \(\parallel (O, \rho) \rightarrow P(\theta) \parallel\), is

\[
\parallel (O, \rho) \rightarrow P(\theta) \parallel (\theta') \equiv tr([P(\theta')]|_{\rho}), \forall \theta' \in \mathbb{R}^k.
\]

Namely, \(\parallel (O, \rho) \rightarrow P(\theta) \parallel\) is a function from \(\mathbb{R}^k\) to \(\mathbb{R}\) whose value per point is given by (5.3).

It’s easy to check that evolving pursuant to the normal programs operational semantics (Fig 1) agrees with \(\parallel P(\theta') \parallel\).

Similarly, for any \(S(\theta) \in add\text{-}q\text{-}while^{T}_\downarrow(\theta)\), Compile\((S(\theta)) = \{P_i(\theta)\}_{i=1}^{l}\) where \(P_i(\theta) \in q\text{-}while^{T}_\downarrow(\theta),\) its observable semantics is given by, \(\forall \theta'^* \in \mathbb{R}^k,\)

\[
\parallel (O, \rho) \rightarrow S(\theta) \parallel (\theta') \equiv \sum_{i \epsilon [1, l]} \parallel (O, \rho) \rightarrow P_i(\theta) \parallel (\theta').
\]

To compute gradients of quantum observables for each parameter, one needs an ancilla variable as hinted by results in quantum information theory about gradient calculations for simple units (e.g., Bergholm et al. [8], Schuld et al. [44]). To that end, we can easily extend quantum programs with ancilla variables. For each \(j \in [1, k],\) the \(j\)-th ancilla of \(q\text{-}while^{T}_\uparrow(\theta)\) is a quantum variable denoted by \(A_j, \tau\) disjoint from \(\bar{V}\). We write \(A\) instead of \(A_j, \tau\) when \(j, \bar{V}\) are clear from context. Ancilla \(A\) could consist of any number of qubits while we will mostly use one-qubit \(A\) in this paper.

We will only consider programs augmented with one ancilla variable \(A_j\) at any time. (So let us fix \(j\) for the following discussion.) We will then consider programs that operate on the larger space \(D(H_{\uparrow} \cup \{\bar{A}\})\) and an additional observable \(A\) to define the observable semantics with ancilla.

**Definition 5.2 (Observable Semantics with Ancilla).** Given any \(P'(\theta) \in q\text{-}while^{T}_\uparrow(\tau_{\downarrow}\cup\{A_j\})(\theta),\) any observable \(O \in O_\uparrow,\) input state \(\rho \in D(H_\uparrow),\) and moreover the observable \(O_A\) on ancilla \(A,\) the observable semantics with ancilla of \(P,\) overloading the notation \(\parallel (O, \rho) \rightarrow P'(\theta) \parallel,\) is

\[
\parallel (O_A \otimes O) \parallel (P'(\theta')) \equiv tr\left((O_A \otimes O) P'(\theta')\right), \forall \theta' \in \mathbb{R}^k.
\]

Again, \(\parallel (O_A \otimes O) \parallel (P'(\theta))\) is a function from \(\mathbb{R}^k\) to \(\mathbb{R}\) whose value per point is given by (5.5).

Similarly, for \(S'(\theta) \in add\text{-}q\text{-}while^{T}_\downarrow(\tau_{\uparrow}\cup\{A_j\})(\theta)\) s.t. Compile\((S'(\theta)) = \{P_i(\theta)\}_{i=1}^{l}\) where \(P_i(\theta) \in q\text{-}while^{T}_\down\downarrow(\tau_{\uparrow}\cup\{A_j\})(\theta),\) its observable semantics is: \(\forall \theta'^* \in \mathbb{R}^k,\)

\[
\parallel (O_A \otimes O) \parallel (S'(\theta')) \equiv \sum_{i \epsilon [1, l]} \parallel (O_A \otimes O) \rightarrow P_i(\theta) \parallel (\theta').
\]

The only difference from the normal observable semantics lies in (5.5), where we initialize the ancilla with \(|0\rangle\), which is a natural choice and evaluate the observable \(O_A \otimes O\). As we will see in the technique, the independence between \(O_A\) and \(O\) in the form of \(O_A \otimes O\) will help us obtain the strongest guarantee of our differentiation procedure.

### 5.2 Differential Semantics

Given the definition of observable semantics, its differential semantics can be naturally defined by
Definition 5.3 (Differential Semantics). Given an additive program \(S(\theta) \in \text{add-q-while}^T_D(\theta)\), its \(j\)-th differential semantics is defined by

\[
\frac{\partial}{\partial \theta_j}((O, \rho) \rightarrow S(\theta)),
\]

which is again a function from \(\mathbb{R}^k \) to \(\mathbb{R}^k\). Moreover, for any \(S'(\theta) \in \text{add-q-while}^T_D(\theta)\), we say that \(S'(\theta)\) computes the \(j\)-th differential semantics of \(S(\theta)\) if and only if there exists an observable \(O_A\) on ancilla \(A\) for \(S'(\theta)\) such that \(\forall O \in O_A, \rho \in \mathcal{D}(\mathcal{H}_\Omega)\),

\[
\frac{\partial}{\partial \theta_j}((O, \rho) \rightarrow S(\theta)).
\]

We remark that (5.6) is well defined because \(\frac{\partial}{\partial \theta_j}((O, \rho) \rightarrow S(\theta))\) is a function from \(\mathbb{R}^k \) to \(\mathbb{R}^k\). It is also a smooth function because we assume that parameterized unitaries are entrywise smooth, and the observable semantics is obtained by multiplication and addition of such entries. Note also that there is one specific choice of \(O_A\) in our current design. We leave it as a parameter to allow flexibility for future designs.

We also remark that the order of quantifiers in (5.7) is the strongest that one can hope for. This is because the observable semantics of \(S(\theta)\) will depend on \(O\) and \(\rho\) in general. Thus, the program to compute its differential semantics could also depend on \(O\) and \(\rho\) in general. However, in our definition, \(S'(\theta)\) is a single fixed program that works for any \(O\) and \(\rho\) regardless of the seemingly complicated relationship. This definition is consistent with the classical case where a single program can compute the derivatives for any input. We can achieve the same definition in the quantum setting and it is critical in the proof of Theorem 6.2 (item (5)).

6 Code Transformations and the Differentiation Logic

We describe the code transformation rules of the differentiation operator \(\frac{\partial}{\partial \theta} \cdot \) in Section 6.1. We also define a logic and prove its soundness for reasoning about the correctness of these code transformations, with the following judgement

\[
S'(\theta) | S(\theta),
\]

which states that \(S'(\theta)\) computes the differential semantics of \(S(\theta)\) in the sense of Definition 5.3. We fix \(\theta = \theta_1\) and hence \(A\) stands for \(A_{\theta_1}\) and \(\frac{\partial}{\partial \theta_1}\) for \(\frac{\partial}{\partial \theta}\) through this section.\(^7\)

6.1 Code Transformations

We first define some gates associated with the single-qubit rotation and the two-qubit coupling gates, which will appear in the code transformation rules. Let \(A\) be a single qubit.

\(\text{(Trivial)}\) \(\frac{\partial}{\partial \theta} (\text{abort}[\mathbb{V}]), \frac{\partial}{\partial \theta} (\text{skip}[\mathbb{V}]), \frac{\partial}{\partial \theta} (q := [0]) \equiv \text{abort}[\mathbb{V} \cup \{A\}].\)

\(\text{(Trivial-U)}\) \(\frac{\partial}{\partial \theta} (U(\mathbb{V})), \frac{\partial}{\partial \theta} (\{q := [0]\}) \equiv \text{abort}[\mathbb{V} \cup \{A\}].\)

\(\text{(1-qb)}\) \(\frac{\partial}{\partial \theta} (q_1 := R_a(\theta)[q_1]) \equiv \{A, q_1 := R'_a(\theta)[A, q_1]\} \equiv \{A, q_1 := R'_a(\theta)[A, q_1, q_2]\}.

\(\text{(2-qb)}\) \(\frac{\partial}{\partial \theta} (q_1, q_2 := R_{a \otimes \sigma}(\theta)[q_1, q_2]) \equiv \{A, q_1 := R'_a(\theta)[A, q_1, q_2]\}.

\(\text{(Sequence)}\) \(\frac{\partial}{\partial \theta} (S_1(\theta)), S_2(\theta)) \equiv (S_1(\theta), \frac{\partial}{\partial \theta} (S_2(\theta))) + (\frac{\partial}{\partial \theta} (S_1(\theta)), S_2(\theta)).

\(\text{(Case)}\) \(\frac{\partial}{\partial \theta} (\text{case } M[\mathbb{V}]) \equiv \text{case } M[\mathbb{V}] \rightarrow \text{end}.\)

\(\text{(while)}\) \(\frac{\partial}{\partial \theta} (\text{while } \theta_1 \rightarrow \text{end}).\)

\(\text{(S-C)}\) \(\frac{\partial}{\partial \theta} (S_1(\theta) + S_2(\theta)) \equiv \frac{\partial}{\partial \theta} (S_1(\theta)) + \frac{\partial}{\partial \theta} (S_2(\theta)).\)

\(\text{Figure 4. Code Transformation Rules. For (1-qb Rotation) and (2-qb Coupling), } (\sigma \in \{X, Y, Z\}), R'_a(\theta), R'_{a \otimes \sigma}(\theta) \text{ are as in Definition 6.1. } \theta_1 \neq \theta \text{ means } \text"the unitary } U(\theta) \text" \text{ trivially uses } \theta_2 \text{ e.g. for example in } P(\theta) \equiv R_X(\theta) ; R_Z(\theta_2), \theta = (\theta_1, \theta_2) \text{ and } R_X(\theta_1) \text{ trivially uses } \theta_2.\)

Definition 6.1. 1. Consider unitary \(R_a(\theta)\) where \(\sigma \in \{X, Y, Z\}\). We define unitary \(C_{R_a}(\theta)\) as

\[
C_{R_a}(\theta) \equiv |0 \rangle \langle 0| \otimes R_a(\theta) + |1 \rangle \langle 1| \otimes R_a(\theta + \pi).
\]

We also define a new gadget program \(R'_a(\theta)\) as

\[
R'_a(\theta)[A, q_1] \equiv A := H[A]; A, q_1 := C_{R_a}(\theta)[A, q_1]; A := H[A].
\]

2. Substituting \(\sigma \otimes \sigma\) for \(\sigma, q_1, q_2, q_1, q_2\) in Eqns (6.2, 6.3), one defines \(C_{R_{a \otimes \sigma}(\theta)}, R'_{a \otimes \sigma}(\theta)\).

\[\text{For 1-qubit rotation } R_a(\theta), \text{ the "controlled-rotation" gate } C_{R_a}(\theta) \text{ maps } |0, q_1| \leftrightarrow |0 \rangle \otimes R_a(\theta)[q_1], \text{ and } |1, q_1| \leftrightarrow |1 \rangle \otimes R_a(\theta + \pi)[q_1]; R'_a(\theta) \text{ conjugates } C_{R_a}(\theta) \text{ with Hadamard. Similarly for corresponding two-qubit coupling gates.}\]

We exhibit our code transformation rules in Figure 4. For Unitary rules we only include 1-qubit rotations and two-qubit coupling gates, since they form a universal gate set and are easy to implement on quantum machines. It is also possible to include more unitary rules (e.g. by following the calculations in [44]), which we will leave as future directions.

6.2 The differentiation logic and its soundness

We develop the differentiation logic given in Figure 5 to reason about the correctness of code transformations. It suffices to show that our logic is sound. For ease of notation, in future analysis we write \(\frac{\partial}{\partial \theta} (P(\theta))\) in place of \(\frac{\partial}{\partial \theta} (P(\theta))\) when \(P(\theta) \in \text{add-q-while}^T_D(\theta)\).

Theorem 6.2 (Soundness). Let \(S(\theta) \in \text{add-q-while}^T_D(\theta), S'(\theta) \in \text{add-q-while}^T_D(\theta), q(\theta) \in \text{add-q-while}^T_D(\theta).\) Then, \(S'(\theta)|S(\theta)\) implies that \(S'(\theta)\) computes the differential semantics of \(S(\theta)\).
(Abort) \( \frac{\partial}{\partial \theta}(\text{abort}[\varnothing]) = \text{abort}[\varnothing] \)

(Skip) \( \frac{\partial}{\partial \theta}(\text{skip}[\varnothing]) = \text{skip}[\varnothing] \)

(Initialized) \( \frac{\partial}{\partial \theta}(q = [0]) = [0] \)

(Trivial-Unitary) \( \frac{\partial}{\partial \theta}(\bar{q} = U(\theta)[\varnothing]) = U(\theta)[\varnothing] \)

(Rot-Couple) \( \frac{\partial}{\partial \theta}(\bar{q} = R_\sigma(\theta)[\varnothing]) = R_\sigma(\theta)[\varnothing] \)

\( \frac{\partial}{\partial \theta}(S_m(\theta)) = S_m(\theta) \)

(Sequence) \( \frac{\partial}{\partial \theta}(S_1(\theta)) = S_1(\theta) \)

(While) \( \frac{\partial}{\partial \theta}([q] = 1 \text{ do } S_1(\theta) \text{ done}) = [q] = 1 \text{ do } S_1(\theta) \text{ done} \)

(Sum Component) \( \frac{\partial}{\partial \theta}(S_1(\theta) + S_1(\theta)) = S_1(\theta) + S_1(\theta) \)

Figure 5. The differentiation logic. Wherever applicable, \( q \in \{0, 1\} \), \( S_1(\theta) \in \text{add-q-while}_{\tau_{\theta, A}}^{(T)}(\theta) \). In (Rot-Couple), \( \sigma \in \{X, Y, Z, X \otimes Y, Y \otimes Z, Z \otimes Z\} \).

Let us highlight the ideas behind the proof of the soundness and all detailed proofs are deferred to Appendix D. First remember that \( \theta = \theta_1 \) and for all the proofs we can choose \( \theta_A = |0\rangle|0\rangle - |1\rangle|1\rangle \) as the observable on the one-qubit ancilla A. Thus, we will omit \( \theta_A \) and overload the notation, \( \forall P(\theta) \in \text{q-while}_{\tau_{\theta, A}}^{(T)}(\theta) \): \( \langle (O, \rho) \rightarrow P(\theta) \rangle \) means \( \langle (O, Z_A, \rho) \rightarrow P(\theta) \rangle \).

(6.4) to simplify the presentation. We make similar overloading convention for \( S'(\theta) \in \text{add-q-while}_{\tau_{\theta, A}}^{(T)}(\theta) \). Let us go through these logic rules one by one.

1. Abort, Skip, Initialization, Trivial-Unitary rules work because these statements do not depend on \( \theta \).
2. Since While \( (T) \) can be deemed as a macro of other statements, the correctness of While \( (T) \) rule follows by unfolding while \( (T) \) and applying other rules.
3. The Sum Component rule is due to the property of observable semantics \( \langle |\rangle \rangle \) and additive operator (+):

\[ \frac{\partial}{\partial \theta} \langle P_1 + P_2 \rangle = \left\langle \frac{\partial}{\partial \theta} P_1 \right\rangle + \left\langle \frac{\partial}{\partial \theta} P_2 \right\rangle \]

(6.5) which follows from our definition design.

Our Rot-Couple rule is different from the phase-shift rule in [44] by using only one circuit in derivative computing. However, the proof of the Rot-Couple rule is largely inspired by the one of the phase-shift rule.

5. The proof of the Sequence rule relies very non-trivially on our design of the observable semantics with ancilla (Definition 5.2) and the strong requirement of computing differential semantics in Definition 5.3. Firstly, note that

\[ \langle (O, \rho) \rightarrow \frac{\partial}{\partial \theta} (S_1(\theta)) \rangle = \langle (O, \rho) \rightarrow \frac{\partial}{\partial \theta} (S_1(\theta)) \rangle \]

We use the induction hypothesis to reason about each term above. Consider the case \( S_1(\theta) = S_1(\theta) \) and \( S_1(\theta) = S_1(\theta) \). Note that \( S_1(\theta), S_1(\theta) \in \text{q-while}_{\tau_{\theta, A}}^{(T)}(\theta) \) and \( \frac{\partial}{\partial \theta}(S_1(\theta)) \in \text{add-q-while}_{\tau_{\theta, A}}^{(T)}(\theta) \). First, show

\[ \langle (O, \rho) \rightarrow S_1(\theta), \theta \rangle = \langle (O, S_1(\theta), \theta) \rangle \rightarrow \frac{\partial}{\partial \theta} (S_1(\theta)) \rangle \]

(6.6)

This is because \( \frac{\partial}{\partial \theta}(S_1(\theta)) \) computes the derivative for any input state and observable. We simply choose the input state \( S_1(\theta) \langle |\rho \rangle \rangle \) and observable \( O \). Secondly, we show

\[ \langle (O, \rho) \rightarrow \frac{\partial}{\partial \theta} (S_1(\theta)) \rangle = \langle (O, S_1(\theta), \theta) \rangle \rightarrow \frac{\partial}{\partial \theta} (S_1(\theta)) \rangle \]

(6.7)

For (6.7), we don't change the state \( \rho \) but change the observable \( O \) by applying the dual super-operator \( \langle S_1(\theta) \rangle \rangle \).

Since \( \frac{\partial}{\partial \theta}(S_1(\theta)) \) computes the derivative for any input state and any observable, we choose the input state \( \rho \) and observable \( \langle S_1(\theta) \rangle \rangle \langle O \rangle \). The dual super-operator \( \langle S_1(\theta) \rangle \rangle \) has the property that \( \text{tr}(O[S_1(\theta) (\rho)]) = \text{tr}([S_1(\theta)]^*(\rho)) \), which corresponds to the Schrodinger picture (evolving states) and Heisenberg picture (evolving observables) respectively in quantum mechanics.

6. The proof of the Case rule basically follows from the linearity of the observable semantics and the smooth semantics of Case. It is interesting to compare with the classical case [6] where the non-smoothness of the guard causes an issue for auto differentiation.

Example 6.1 (Simple-Case). Consider the following simple instantiating of Example 4.1

\[ P(\theta) \triangleq \text{case } q_1 \rightarrow \begin{array}{ll}
R_X(\theta)q_1; & R_Y(\theta)q_1, \\
R_Z(\theta)q_1, &
\end{array} \]

Let us apply code transformation and compilation. Let CT, CP to denote "code transformation" and "compilation", and "Seq" and "Rot" denote Sequence and Rotation rules resp.

\[ \frac{\partial}{\partial \theta} (P(\theta)) \begin{array}{l}
\text{case } q_1 \rightarrow \begin{array}{ll}
\frac{\partial}{\partial \theta} (R_X(\theta)q_1); & \frac{\partial}{\partial \theta} (R_Y(\theta)q_1), \\
\frac{\partial}{\partial \theta} (R_Z(\theta)q_1); &
\end{array}
\end{array} \]

(6.8)
7 Execution and Resource Analysis

In this section we illustrate the execution of the entire differentiation procedure and analyze its resource cost. Consider any program $P(\theta) \in \mathcal{Q}$-while$_{||}^T(\theta)$ and the parameter $\theta$.

**Execution.** The first step in differentiation is to apply the code transformation rules (in Section 6) to $P(\theta)$ and obtain an additive program $\frac{\partial}{\partial \theta}(P(\theta))$. Then one needs to compile $\frac{\partial}{\partial \theta}(P(\theta))$ into a multiset $\{P'_i(\theta)\}_{i=1}^m$ of normal non-aborting programs $P'_i(\theta)$. The total count of these programs is given by $m = \# \frac{\partial}{\partial \theta}(P(\theta))$. Note that the above procedure could be done at the compilation time.

Given any pair of $O$ and $\rho$, the real execution to compute the derivative of $\{ (O, \rho) \rightarrow P(\theta) \}$ is to approximate the observable semantics $\{ (O, \rho) \rightarrow \frac{\partial}{\partial \theta}(P(\theta)) \}$. By Definition 5.2, we need to approximate

$$\sum_{i=1}^m \text{tr} \left( (Z_A \otimes O) \cdot [P'_i(\theta) \cdot (\langle \theta | A \rangle \langle A | \theta \rangle) \otimes \rho] \right),$$

where each term is the observable $Z_A \otimes O$ on the output state of $P'_i(\theta)$ given input state $\rho$ and the ancilla qubit $|0\rangle$.

To approximate the sum in (7.1) to precision $\delta$, one could first treat the sum divided by $m$ as the observable applied on the program that starts with a uniformly random choice of $i$ from $1, \cdots, m$ and then execute $P'_i(\theta)$. By Chernoff bound, one only needs to repeat this procedure $O(m^2/\delta^2)$ times.

**Resource count.** We are only interested in non-trivial (extra) resource that is something that you wouldn’t need if you only run the original program. Ancilla qubits count as the non-trivial resource. However, for our scheme, the number of required ancillae is 1 qubit per each parameter.

The more non-trivial resource is the number of the copies of input state (each copy of the input state is to be prepared from scratch), which is directly related to the number of repetitions in the procedure, which again connects to $m = \# \frac{\partial}{\partial \theta}(P(\theta))$. We argue that our code transformation is efficient so that $m$ is reasonably bounded. To that end, we show the relation between $m$ and a natural quantity defined on the original program $P(\theta)$ (i.e., before applying any $\frac{\partial}{\partial \theta}(-)$ operator) called the occurrence count of the parameter $\theta$.

**Definition 7.1.** The “Occurrence Count for $\theta_j$” in $P(\theta)$, denoted $OC_j(P(\theta))$, is defined as follows:

1. If $P(\theta) \equiv \text{abort} || \text{skip} || q : = \langle 0 \rangle$, then $OC_j(P(\theta)) = 0$;
2. $P(\theta) \equiv U(\theta)$: if $U(\theta)$ trivially uses $\theta_j$, then $OC_j(P(\theta)) = 0$; otherwise $OC_j(P(\theta)) = 1$.
3. If $P(\theta) \equiv U(\theta) = P_1(\theta); P_2(\theta)$ then $OC_j(P(\theta)) = OC_j(P_1(\theta)) + OC_j(P_2(\theta))$.
4. If $P(\theta) \equiv \text{case } M[\overline{q}] = \overline{m} \rightarrow \overline{P}_m(\theta)$ end then $OC_j(P(\theta)) = \max_m OC_j(P_m(\theta))$.
5. If $P(\theta) \equiv \text{while}^T M[\overline{q}] \equiv 1 \text{ do } P_1(\theta) \text{ done }$ then $OC_j(P(\theta)) = T \cdot OC_j(P_1(\theta))$.

Intuition of the “Occurrence Count” definition is clear: it basically counts the number of non-trivial occurrences of $\theta_j$ in the program, treating case as if it is deterministic. To see why this is a reasonable quantity, consider the auto-differentiation in the classical case. For any non-trivial variable $v$ (i.e., $v$ has some dependence on the parameter $\theta_j$), we will compute both $v$ and $\frac{\partial}{\partial \theta_j}(v)$ and store them both as variables in the new program. Thus, the classical auto-differentiation essentially needs the number of non-trivial occurrences more space and related resources. As we argued in the introduction, we cannot directly mimic the classical case due to the no-cloning theorem. The extra space requirement in the classical setting turns into the requirement on the extra copies of the input state in the quantum setting. Indeed, we can bound $m$ by the occurrence count.

**Proposition 7.2.** $\# \frac{\partial}{\partial \theta_j}(P(\theta)) \leq OC_j(P(\theta))$.

**Proof.** Structural induction. For details, see Appendix E.1.

8 Implementation and Case Study

We have built a compiler (written in OCaml) that implements our code transformation and compilation rules\(^a\). We use it to train one VQC instance with controls and empirically verify its resource-efficiency on representative VQC instances. Complete details can be found in Appendix F. Experiments are performed on a MacBook Pro with a Dual-Core Intel Core i5 Processor clocked at 2.7 GHz, and 8GB of RAM.

8.1 Training VQC instances with controls

Consider a simple classification problem over 4-bit inputs $z = z_1 z_2 z_3 z_4 \in \{0, 1\}^4$ with true label given by $f(z) = \neg (z_1 \oplus z_4)$. We construct two 4-qubit VCQCs $P_1$ (no control) and $P_2$ (with control) that consists of a single-qubit Pauli $X,Y$ and $Z$ rotation gate on each qubit and compare their performance.

For parameters $\Gamma = \{ y_1, \ldots, y_{12} \}$ define the program

$$Q(\Gamma) \equiv \begin{array}{l}
R_Z(y_1)[q_1]; R_X(y_1)[q_2]; R_X(y_1)[q_3]; R_X(y_1)[q_4]; \\
R_Y(y_5)[q_1]; R_Y(y_6)[q_2]; R_Y(y_7)[q_3]; R_Y(y_8)[q_4]; \\
R_Z(y_9)[q_1]; R_Z(y_{10})[q_2]; R_Z(y_{11})[q_3]; R_Z(y_{12})[q_4].
\end{array}$$

\(^a\)Codes are available at https://github.com/HeritasSpZ/adcompile.
where \(q_1, q_2, q_3, q_4\) refer to 4 qubit registers. Given parameters \(\Theta = \{\theta_1, \ldots, \theta_{12}\}, \Phi = \{\phi_1, \ldots, \phi_{12}\}\), define
\[
P_1(\Theta, \Phi) \equiv Q(\Theta); Q(\Phi). \tag{8.1}
\]
Similarly, for parameters \(\Theta = \{\theta_1, \ldots, \theta_{12}\}, \Phi = \{\phi_1, \ldots, \phi_{12}\}, \Psi = \{\psi_1, \ldots, \psi_{12}\}\), define
\[
P_2(\Theta, \Phi, \Psi) \equiv Q(\Theta); M[q_1 = 0 \rightarrow Q(\Phi), 1 \rightarrow Q(\Psi)]. \tag{8.2}
\]
Note that \(P_1\) and \(P_2\) execute the same number of gates for each run. To use \(P_1\) to perform the classification or in the training, we first initialize \(q_1, q_2, q_3, q_4\) to the classical feature vector \(z = z_1z_2z_3z_4\) and then execute \(P_1\). The predicted label \(y\) is given by measuring the 4th qubit \(q_4\) in the 0/1 basis.

We conduct a supervised learning by minimizing a loss function. A natural choice is the average negative log-likelihood which is commonly used in machine learning to evaluate classifiers that assign a certain probability to each label since quantum outcomes are probabilistic. However, this loss function is not currently supported by Pennyyle. Denote the output of the classifier with input \(z\) and parameters \(\theta\) by \(l_\theta(z)\). To enable a direct comparison, we will treat \(l_\theta(z)\) as the average value of the labels from probabilistic quantum outcomes, and use the squared loss function as follows:
\[
\text{loss} = \sum_{z \in \{0, 1\}^4} 0.5 \times (l_\theta(z) - f(z))^2. \tag{8.3}
\]
Note that loss is a function of \(\theta = (\Theta, \Phi)\) or \((\Theta, \Phi, \Psi)\). More importantly, for each \(z\), \(l_\theta(z)\) can be represented by the observable semantics of \(P_1\) (or \(P_2\)) with observable \(|1\rangle\langle 1|\). Thus, the gradient of loss can be obtained by using the collection of \(\frac{\partial}{\partial \alpha} (P_1)\) for \(\alpha \in \Theta, \Phi\) (or \(\frac{\partial}{\partial \alpha} (P_2)\) for \(\alpha \in \Theta, \Phi, \Psi\)). We classically simulate the training procedure with gradient descent. For the training of \(P_1\), we use Pennyyle for a direct comparison (see Figure 6). After 1000 epochs with some hyperparameters, the loss for \(P_1\) (no control) attains a minimum of 0.5 in less than 100 epochs and subsequently plateaus. The loss for \(P_2\) (with control) continues to decrease and attains a minimum of 0.016. It demonstrates the advantage of both controls in quantum machine learning and our scheme to handle controls, whereas previous schemes (such as Pennyyle due to its quantum-node design [8]) fail to do so.

### 8.2 Benchmark testing on representative VQCs

We also test our compiler on important VQC candidates such as quantum neural-networks (QNN) for solving machine learning tasks [18], quantum approximate optimization algorithms (QAOA) for solving combinatorial optimization [17], and variational quantum eigensolver (VQE) for approximating ground state energies in quantum chemistry [38], all of which are promising candidates for actual implementation on near-term quantum machines. These VQCs typically consists of alternating layers of single-qubit gates and two-qubit coupling gates, such as the 1-qubit, 2-qubit Pauli rotation gates considered in our paper, to represent the alternation between local interaction and neighboring interaction in real quantum physics systems.

We enrich these examples, by adding simple controls (the \(\text{if/condition}\) statement) or 2-bounded loops (the \(\text{bounded-while}\) statement) and increasing the number of qubits to 18–40, to make them sufficiently sophisticated but yet realistic for near-term quantum applications. For example, we use \(\text{QNN}\) to denote an enriched QNN VQC instance of medium size and with if controls. The size of \(\text{QNN}_{M,i}\) can also be directly illustrated by the number of qubits (#qubits), the gate count (#gates), the number of alternating layers (#layers), and the number of lines to code such instances (#lines). Similarly for \(\text{QNN}_{L,w}\) except that it is an instance of large size and with while controls.

A selective output performance of our compiler is in Table 2, with details in Appendix F. It is easy to see that our scheme is also empirically resource-efficient as \(\# \frac{\partial}{\partial \theta} (\cdot)\) is always reasonably bounded.

![Figure 6. Training \(P_1\) and \(P_2\) to classify inputs according to the labelling function \(f(z) = -(z_1 \oplus z_4)\).](image)

| \(P(\theta)\) | OC(\cdot) | \# gates | \# lines | \# layers | \# qubits |
|---|---|---|---|---|---|
| \(\text{QNN}_{M,i}\) | 24 | 24 | 165 | 189 | 18 |
| \(\text{QNN}_{M,w}\) | 56 | 24 | 231 | 121 | 18 |
| \(\text{QNN}_{L,i}\) | 48 | 48 | 363 | 414 | 36 |
| \(\text{QNN}_{L,w}\) | 504 | 48 | 2079 | 244 | 36 |
| \(\text{VQE}_{M,i}\) | 15 | 15 | 224 | 241 | 12 |
| \(\text{VQE}_{M,w}\) | 35 | 15 | 224 | 112 | 12 |
| \(\text{VQE}_{L,i}\) | 40 | 40 | 576 | 628 | 40 |
| \(\text{VQE}_{L,w}\) | 248 | 40 | 1984 | 368 | 40 |
| \(\text{QAOA}_{M,i}\) | 18 | 18 | 120 | 142 | 18 |
| \(\text{QAOA}_{M,w}\) | 42 | 18 | 168 | 94 | 18 |
| \(\text{QAOA}_{L,i}\) | 36 | 36 | 264 | 315 | 36 |
| \(\text{QAOA}_{L,w}\) | 378 | 36 | 1512 | 190 | 36 |

Table 2. Output on selective examples. \(\{M, L\}\) stands for "medium, large"; \(\{i, w\}\) stands for including "if, while".
References

[1] Martin Abadi and Gordon D. Plotkin. 2019. A Simple Differentiable Programming Language. *Proc. ACM Program. Lang.* 4, POPL, Article 38 (Dec. 2019), 28 pages. https://doi.org/10.1145/3371106

[2] Ali Javadi Abhari, Arvin Faruque, Mohammad Javad Dousti, Lukas Svec, Oana Catu, Amlan Chakrabati, Chen-Fu Chiang, Seth Vanderbilt, John Black, Fred Chong, Margaret Martonosi, Martin Suchara, Ken Brown, Massoud Pedram, and Todd Brun. 2012. *Scaffold: Quantum Programming Language.* Technical Report TR-934-12. Princeton University.

[3] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Supak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergeï V. Isakov, Evan Jeffrey, Zhang Ji, Divr Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landsile, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megnrat, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naiman, Matthew Neeley, Charles Neill, Murphy Yuehzen Niu, Eric Ostby, Andre Petukhov, John Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Sattlinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trottivist, Adam Ainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510.

[4] Alexandru Baltag and Sonja Smets. 2011. Quantum Logic as a Dynamic Logic. *Synthese* 179, 2 (2011).

[5] Atılım Günes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Alexandru Baltag and Sonja Smets. 2011. Quantum Logic as a Dynamic Logic. In *Quantum Information and Inference.* Springer, 1–49. https://doi.org/10.1007/978-3-642-10622-4_7

[6] Thomas Beck and Herbert Fischer. 1994. The if-problem in automatic differentiation. *J. Comput. Appl. Math.* 50, 1-3 (1994), 119–131.

[7] Marcello Benedetti, Erika Lloyd, and Stefan Sack. 2019. Parameterized quantum circuits as machine learning models. *arXiv e-prints* (Jun 2019). arXiv:1906.07682

[8] Ville Bergholm, Josh Izac, Maria Schuld, Christian Gogolin, and Nathan Killoran. 2018. PennyLane: Automatic differentiation of hybrid quantum-classical computations. *arXiv:1811.04968* (2018).

[9] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. 2017. Quantum machine learning. *Nature* 549, 7671 (2017), 195.

[10] Olivier Brunet and Philippe Jorrand. 2004. Dynamic Quantum Logic for Quantum Programs. *International Journal of Quantum Information* 2, 1 (2004).

[11] Rohit Chadha, Paulo Mateus, and Amilcar Sernadas. 2006. Reasoning About Imperative Quantum Programs. *Electronically Notes in Theoretical Computer Science* 158 (2006).

[12] George Corliss, Christèle Faure, Andreas Griewank, Lauren Hascoët, and Uwe Naumann (Eds.). 2002. *Automatic Differentiation of Algorithms: From Simulation to Optimization.* Springer-Verlag New York, Inc., New York, NY, USA.

[13] Ellie D’Hondt and Prakash Panangaden. 2006. Quantum Weakest Preconditions. *Mathematical Structures in Computer Science* 16, 3 (2006).

[14] Thomas Ehrhard and Laurent Regnier. 2003. The differential lambda-calculus. *Theoretical Computer Science* 309, 1-3 (2003), 1–41.

[15] Conal M. Elliott. 2009. Beautiful Differentiation. In *Proceedings of the 14th ACM SIGPLAN International Conference on Functional Programming* (Edinburgh, Scotland) (ICFP ’09). ACM, New York, NY, USA, 191–202. https://doi.org/10.1145/1596560.1596579

[16] Conal M. Elliott. 2018. The Simple Essence of Automatic Differentiation. *Proc. ACM Program. Lang.* 2, ICFP, Article 70 (July 2018), 29 pages. https://doi.org/10.1145/3236765

[17] Edward Farhi, Jeffrey Gelladstone, and Sam Gutmann. 2014. A Quantum Approximate Optimization Algorithm. (2014). arXiv:1411.0402

[18] Edward Farhi and Hartmut Neven. 2018. Classification with Quantum Neural Networks on Near Term Processors. (2018). arXiv:1802.06002

[19] Yuan Feng, Runyao Duan, Zhengfeng Ji, and Mingsheng Ying. 2007. Proof Rules for the Correctness of Quantum Programs. *Theoretical Computer Science* 386, 1-2 (2007).

[20] Simon J. Gay. 2006. Quantum Programming Languages: Survey and Bibliography. *Mathematical Structures in Computer Science* 16, 4 (2006).

[21] Gian Giacomo Guerreschi and Mikhail Smelyanskiy. 2017. Practical optimization for hybrid quantum-classical algorithms. (2017). arXiv:1701.01450

[22] Martin Giles. 2019. IBM’s new 53-qubit computer is the most powerful machine you can use. https://www.technologyreview.com/614346/ibm-new-53-qubit-quantum-computer-is-the-most-powerful-machine-you-can-use/

[23] Pranav Gokhale. 2018. Variational Quantum Eigensolver Demo. ISCA 2018 (2018).

[24] Jonathan Grattage. 2005. A Functional Quantum Programming Language. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS ’05).* IEEE Computer Society, USA, 2497258. https://doi.org/10.1109/LICS.2005.1

[25] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènec Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature* 538 (10 2016), 471.

[26] Alexander S. Green, Peter LeFaux Lumsdaine, Neil J. Ross, Peter Selinger, and Benoit Valiron. 2013. Quipper: A Scalable Quantum Programming Language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Seattle, Washington, USA) (PLDI ’13). Association for Computing Machinery, New York, NY, USA, 334–343. https://doi.org/10.1145/2491956.2462177

[27] Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to Transduce with Unbounded Memory. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2* (Montreal, Canada) (NIPS ’15). MIT Press, Cambridge, MA, USA, 1828–1836. http://dl.acm.org/citation.cfm?id=2964942.2964944

[28] Andreas Griewank. 2000. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation,* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[29] Shih-Han Hung, Keshu Hietala, Shaopeng Zhu, Mingsheng Ying, Michael Hicks, and Xiaodong Wu. 2019. Quantitative Robustness Analysis of Quantum Programs. *Proc. ACM Program. Lang.* 3, POPL, Article 31 (Jan. 2019), 29 pages. https://doi.org/10.1145/3290344

[30] Yoshihiko Kudoh. 2009. A Logic for Formal Verification of Quantum Programs. In *Proceedings of the 13th Asian Conference on Advances in Computer Science: Information Security and Privacy* (Seoul, Korea) (ASIAN ’09). Springer-Verlag, Berlin, Heidelberg, 79–93. https://doi.org/10.1007/978-3-642-10622-4_7

[31] Gershon Kedem. 1980. *Automatic Differentiation of Computer Programs.* *ACM Trans. Math. Softw.* 6, 2 (June 1980), 150–165. https://doi.org/10.1145/355887.355890
A Detailed Quantum Preliminary

This is a more detailed treatment of Section 2. For a further extended background, we recommend the notes by Watrous [51] and the textbook by Nielsen and Chuang [34].

A.1 Preliminaries

For any non-negative integer \( n \), an \( n \)-dimensional Hilbert space \( \mathcal{H} \) is essentially the space \( \mathbb{C}^n \) of complex vectors. We use Dirac’s notation, \( |\psi\rangle \), to denote a complex vector in \( \mathbb{C}^n \). The inner product of two vectors \( |\psi\rangle \) and \( |\phi\rangle \) is denoted by \( \langle \psi|\phi \rangle \), which is the product of the Hermitian conjugate of \( |\psi\rangle \), denoted by \( \langle \psi| \), and vector \( |\phi\rangle \). The norm of a vector \( |\psi\rangle \) is denoted by \( \| |\psi\rangle \| = \sqrt{\langle \psi|\psi \rangle} \).

We define (linear) operators as linear mappings between Hilbert spaces. Operators between \( n \times n \) Hilbert spaces are represented by \( n \times n \) matrices. For example, the identity operator \( I_{\mathcal{H}} \) can be identified by the identity matrix on \( \mathcal{H} \). The Hermitian conjugate of operator \( A \) is denoted by \( A^\dagger \). Operator \( A \) is Hermitian if \( A = A^\dagger \). The trace of an operator \( A \) is the sum of the entries on the main diagonal, i.e., \( \text{tr}(A) = \sum_i A_{ii} \). We write \( \langle \psi| A |\psi \rangle \) to mean the inner product between \( |\psi\rangle \) and \( A |\psi \rangle \). A Hermitian operator \( A \) is a positive semidefinite (resp., positive definite) if for all vectors \( |\psi\rangle \in \mathcal{H} \), \( \langle \psi| A |\psi \rangle \geq 0 \) (resp., \( > 0 \)). This gives rise to the Löwner order \( \subseteq \) among operators:

\[
A \subseteq B \quad \text{if} \quad B - A \text{ is positive semidefinite},
\]

\[
A \subsetneq B \quad \text{if} \quad B - A \text{ is positive definite}.
\]

A.2 Quantum States

The state space of a quantum system is a Hilbert space. The state space of a qubit, or quantum bit, is a 2-dimensional Hilbert space. One important orthonormal basis of a qubit system is the computational basis with \( |0\rangle = (1, 0)^\dagger \) and \( |1\rangle = (0, 1)^\dagger \), which encode the classical bits 0 and 1 respectively. Another important basis, called the + basis, consists of \( |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \) and \( |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \). The state space of multiple qubits is the tensor product of single qubit state spaces. For example, classical 00 can be encoded by \( |0\rangle \otimes |0\rangle \) (written \( |00\rangle \) or even \( |00\rangle \) for short) in the Hilbert space \( \mathbb{C}^2 \otimes \mathbb{C}^2 \). An important 2-qubit state is the EPR state \( |\beta_{00}\rangle \equiv \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \). The Hilbert space for an \( m \)-qubit system is \( (\mathbb{C}^2)^\otimes m \equiv \mathbb{C}^{2^m} \).

A pure quantum state is represented by a unit vector, i.e., a vector \( |\psi\rangle \) with \( \| |\psi\rangle \| = 1 \). A mixed state can be represented by a classical distribution over an ensemble of pure states \( \{(p_i, |\psi_i\rangle)\}_i \), i.e., the system is in state \( |\psi_i\rangle \) with probability \( p_i \). One can also use \textit{density operators} to represent both pure and mixed quantum states. A density operator \( \rho \) for a mixed state representing the ensemble \( \{(p_i, |\psi_i\rangle)\}_i \) is a positive semidefinite operator \( \rho = \sum_i p_i |\psi_i\rangle \langle \psi_i| \). The outer product of \( |\psi_i\rangle \langle \psi_i| \) can be identified with the density operator \( \rho = |\psi\rangle \langle \psi| \). Note that \( \text{tr}(\rho) = 1 \) holds for all density operators. A positive semidefinite operator \( \rho \) on \( \mathcal{H} \) is said to be a \textit{partial} density operator if \( \text{tr}(\rho) \leq 1 \). The set of partial density operators is denoted by \( \mathcal{D}(\mathcal{H}) \).

A.3 Quantum Operations

Operations on quantum systems can be characterized by unitary operators. Denoting the set of linear operators on \( \mathcal{H} \) as \( \mathcal{L}(\mathcal{H}) \), an operator \( U \in \mathcal{L}(\mathcal{H}) \) is unitary if its Hermitian conjugate is its own inverse, i.e., \( U^\dagger U = U U^\dagger = I_{\mathcal{H}} \). For a pure state \( |\psi\rangle \), a unitary operator describes an evolution from \( |\psi\rangle \) to \( U |\psi\rangle \). For a density operator \( \rho \), the corresponding evolution is \( \rho \mapsto U \rho U^\dagger \). Common unitary operators include

\[
H = \begin{bmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}
\end{bmatrix},
X = \begin{bmatrix}
0 & 1 \\
1 & 0
\end{bmatrix},
Z = \begin{bmatrix}
1 & 0 \\
0 & -1
\end{bmatrix},
Y = -iXZ.
\]

The Hadamard operator \( H \) transforms between the computational and the ± basis. For example, \( H |0\rangle = |+\rangle \) and \( H |1\rangle = |-\rangle \). The Pauli \( X \) operator is a bit flip, i.e., \( X |0\rangle = |1\rangle \) and \( X |1\rangle = |0\rangle \). The Pauli \( Z \) operator is a phase flip, i.e., \( Z |0\rangle = |0\rangle \) and \( Z |1\rangle = -|1\rangle \). Pauli \( Y \) maps \( |0\rangle \) to \( |1\rangle \) and \( |1\rangle \) to \( -|0\rangle \). The CNOT gate \( C \) maps \( |00\rangle \mapsto |00\rangle, |01\rangle \mapsto |01\rangle, |10\rangle \mapsto |11\rangle, |11\rangle \mapsto |10\rangle \). One may obtain the EPR state \( |\beta_{00}\rangle \equiv |00\rangle \mapsto \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle) \).

More generally, the evolution of a quantum system can be characterized by an \textit{admissible superoperator} \( \mathcal{E} \), which is a completely-positive and trace-non-increasing linear map from \( \mathcal{D}(\mathcal{H}) \) to \( \mathcal{D}(\mathcal{H'}) \) for Hilbert spaces \( \mathcal{H}, \mathcal{H'} \). A superoperator is positive if it maps from \( \mathcal{D}(\mathcal{H}) \) to \( \mathcal{D}(\mathcal{H'}) \) for Hilbert spaces \( \mathcal{H}, \mathcal{H'} \). A superoperator \( \mathcal{E} \) is positive if \( \mathcal{E}(\rho) \) is positive for any \( \rho \in \mathcal{D}(\mathcal{H}) \). A superoperator \( \mathcal{E} \) is a unitary evolution if \( \mathcal{E}(\rho) \) is positive for any positive integer \( k \). A unitary evolution \( \mathcal{E} \) is trace-non-increasing if for any initial state \( \rho \in \mathcal{D}(\mathcal{H}) \), the final state \( \mathcal{E}(\rho) \in \mathcal{D}(\mathcal{H'}) \) after applying \( \mathcal{E} \) satisfies \( \text{tr}(\mathcal{E}(\rho)) \leq \text{tr}(\rho) \).

For every superoperator \( \mathcal{E} : \mathcal{D}(\mathcal{H}) \rightarrow \mathcal{D}(\mathcal{H'}) \), there exists a set of Kraus operators \( \{E_k\}_k \) such that \( \mathcal{E}(\rho) = \sum_k E_k \rho E_k^\dagger \) for any input \( \rho \in \mathcal{D}(\mathcal{H}) \). Note that the set of Kraus operators is finite if the Hilbert space is finite-dimensional. The \textit{Kraus form} of \( \mathcal{E} \) is written as \( \mathcal{E} = \sum_k E_k \otimes E_k^\dagger \). A unitary evolution can be represented by the superoperator \( \mathcal{E} = U \otimes U^\dagger \). An identity operation refers to the superoperator \( I_{\mathcal{H}} \equiv I_{\mathcal{H}} \otimes I_{\mathcal{H}} \). The Schrödinger-Heisenberg \textit{dual} of a superoperator \( \mathcal{E} = \sum_k E_k \otimes E_k^\dagger \) is denoted by \( \mathcal{E}^* \), is defined as follows: for every state \( \rho \in \mathcal{D}(\mathcal{H}) \) and any operator \( A \), \( \text{tr}(\mathcal{E}(\rho) A) = \text{tr}(\mathcal{E}^*(A) \rho) \).

The Kraus form of \( \mathcal{E}^* \) is \( \sum_k E_k^\dagger \otimes E_k \).

A.4 Quantum Measurements and Observables

The way to extract information about a quantum system is called a \textit{quantum measurement}. A quantum measurement on a system over Hilbert space \( \mathcal{H} \) can be described by a set of linear operators \( \{M_m\}_m \) with \( \sum_m M_m^\dagger M_m = I_{\mathcal{H}} \). If we perform a
measurement \(\{M_m\}_m\) on a state \(\rho\), the outcome \(m\) is observed with probability \(p_m = \text{tr}(M_mpM_m^\dagger)\) for each \(m\). A major difference between classical and quantum computation is that a quantum measurement changes the state. In particular, after a measurement yielding outcome \(m\), the state collapses to \(M_m\rho M_m^\dagger/M_m\). For example, a measurement in the computational basis is described by \(M = |0\rangle\langle 0|, M_1 = |1\rangle\langle 1|\).

If we perform the computational basis measurement \(M\) on state \(\rho = |+\rangle\langle +|\), then with probability \(\frac{1}{2}\) the outcome is 0 and \(\rho\) becomes \(|0\rangle\langle 0|\). With probability \(\frac{1}{2}\) the outcome is 1 and \(\rho\) becomes \(|1\rangle\langle 1|\).

**B More on the definition of parameterized quantum programs**

**B.1 Definition of qVar**

Given \(P(\theta)\) a \(T\)-bounded \(k\)-parameterized quantum while-program, let \(\text{qVar}(P(\theta))\), read the set of quantum variables accessible to \(P\), be recursively defined as follows [56]:

1. If \(P(\theta) \equiv \text{skip}[\overline{q}]\), \(\text{abort}[\overline{q}]\) or \(U[\overline{q}]\), then \(\text{qVar}(P(\theta)) = \overline{q}\).
2. If \(P(\theta) = q := |0\rangle\), then \(\text{qVar}(P(\theta)) = q\).
3. If \(P(\theta) \equiv P_1(\theta); P_2(\theta)\), then \(\text{qVar}(P(\theta)) = \text{qVar}(P_1(\theta)) \cup \text{qVar}(P_2(\theta))\). When analyzing \(P_1(\theta)\); \(P_2(\theta)\), we identify \(P_1(\theta)\) with \(\overline{I} \otimes P_1(\theta)\), where \(\overline{I} \equiv \overline{I} \text{qVar}(P_1(\theta)) \otimes \overline{I} \text{qVar}(P_2(\theta))\), the identity operation on the variables where \(P_1\) originally has no access to.
4. If \(P(\theta) \equiv \text{case} M[\overline{q}] = m \rightarrow P_m(\theta)\) end, then \(\text{qVar}(P(\theta)) = \overline{q} \cup \bigcup_m \text{qVar}(P_m(\theta))\). We make the same identification for \(P_m(\theta)\) as the above.
5. If \(P(\theta) \equiv \text{while}^{(T)} M[\overline{q}] = 1\) do \(P_1(\theta)\) done, then \(\text{qVar}(P(\theta)) = \overline{q} \cup \bigcup_q \text{qVar}(P_1(\theta))\). We make the same identification for \(P_1(\theta)\) as the above.

One defines \(\text{qVar}(P)\) for unparameterized \(P\) analogously.

**C Detailed Proof from Section 4**

**C.1 Proof of Prop 4.2: Non-deterministic Compilation Rules Well-Defined**

**Proof:** Structural induction.

1. (Atomic): as operational semantics of atomic operations are inherited from parameterized quantum while programs, they do not induce non-determinism. For these operations, \(\text{Compile}(P(\theta))|\rho\rangle = \{\{\text{Compile}(P(\theta))|\rho\rangle\}\}\) as is the right hand side.
2. (Sequence) If one of \(\text{Compile}(P_k(\theta'))\) is \{\text{abort}\} the statement is immediately true, so we assume otherwise for below.

   a. \(\subseteq\): Let \(\rho' \in \{\rho' \neq 0 : \langle P_1(\theta'); P_2(\theta'), \rho \rangle \rightarrow^* \langle \downarrow, \rho' \rangle\}\). By definition there exists \(\rho_2 \neq 0\) s.t. \(\langle P_1(\theta'); P_2(\theta'), \rho \rangle \rightarrow^* \langle P_2(\theta'), \rho_2 \rangle\) and \(\langle P_2(\theta'), \rho_2 \rangle \rightarrow^* \langle \downarrow, \rho' \rangle\). By inductive hypothesis, \(\rho' \in \bigcup_{Q(\theta') \in \text{Compile}(P_1(\theta'))} \{\rho' \neq 0 : \langle Q(\theta'), \rho_2 \rangle \rightarrow^* \langle \downarrow, \rho' \rangle\}\) for some \(\rho_2\).

   Since \(\langle P_1(\theta'); P_2(\theta'), \rho \rangle \rightarrow^* \langle P_2(\theta'), \rho_2 \rangle\), there must be some \(\langle P_3(\theta'; \rho_3) \rightarrow \langle \downarrow, \rho_2 \rangle\). By inductive hypothesis we have \(\rho_2 \in \bigcup_{R_i(\theta') \in \text{Compile}(P_2(\theta'))} \{\rho' \neq 0 : \langle R_i(\theta'), \rho \rangle \rightarrow^* \langle \downarrow, \rho' \rangle\}\). Thus

\[
\rho' = \bigcup_{R_i(\theta') \in \text{Compile}(P_2(\theta'))} \{\rho' \neq 0 : \langle R_i(\theta'), \rho \rangle \rightarrow^* \langle \downarrow, \rho' \rangle\}
\]

as desired.

b. \(\supseteq\): Let \(\rho'\) be a member of the multiset in right hand side ("RHS") of C.5 for some \(R_i(\theta') \in \text{Compile}(P_1(\theta'))\), \(Q_i(\theta') \in \text{Compile}(P_2(\theta'))\). Then by inductive hypothesis, \(\langle P_1(\theta'), [R_i(\theta')]|\rho\rangle \rightarrow^* \langle \downarrow, \rho' \rangle\) \((\rho' \neq 0)\), and \(\langle P_2(\theta'), \rho \rangle \rightarrow^* \langle \downarrow, [R_i(\theta')]|\rho\rangle\) \((\|R_i(\theta')\|\rho \neq 0)\). One may start with \(\langle P_2(\theta'), [R_i(\theta')]|\rho\rangle \rightarrow^* \langle \downarrow, \rho' \rangle\) then repeatedly apply the \(\text{Sequential rule in the operational semantics, tracing back the path}\) \(\langle P_1(\theta), \rho \rangle \rightarrow^* \langle \downarrow, [R_i(\theta')]|\rho\rangle\), and conclude that \(\langle P_2(\theta'), [R_i(\theta')]|\rho\rangle \rightarrow^* \langle \downarrow, \rho' \rangle\)

The above said the two multisets have the same set of distinct elements, and furthermore each copy of \(\rho' \in \|P(\theta')\|\rho\) induces at least (by \(\subseteq\), or by the deterministic nature of \(R_i(\theta')\); \(Q_i(\theta')\)) as well as at most (by \(\supseteq\)) one copy of the same element in RHS(C.5). This says the two multisets are equal.

3. (Case) Applying the operational transition rule for 1 step one may observe that (writing "IH" the inductive hypothesis)

\[
\begin{align*}
\{0 \neq \rho' \in \text{Compile}(P(\theta')) & \rightarrow^* \langle \downarrow, \rho' \rangle\} \\
= \bigcup_{m'} \{0 \neq \rho' : \langle P_m(\theta'), M_m\rho M_m^\dagger \rangle \rightarrow \langle \downarrow, \rho' \rangle\} \\
& \quad \text{IH} \\
\end{align*}
\]

\[
\begin{align*}
\{0 \neq \rho' \in \text{Compile}(P(\theta')) & \rightarrow^* \langle \downarrow, \rho' \rangle\} \\
= \bigcup_{m'} \{0 \neq \rho' : \langle Q_m(\theta'), M_m\rho M_m^\dagger \rangle \rightarrow^* \langle \downarrow, \rho' \rangle\} \\
\end{align*}
\]

\footnote{We don’t count "transitions" like \(\langle P(\theta'), \rho \rangle \rightarrow \langle P(\theta'), \rho \rangle\) when analyzing computation paths, as nothing evolves in these "trivial transitions".}
On the other hand we have

\[
\begin{align*}
Q(\theta) & \equiv \text{Compile}_{\text{case}}(M[\overline{\theta}] \rightarrow P_m(\theta), \end{align*}
\]

\[
\{(\rho' \neq 0 : \langle Q(\theta'), \rho \rangle \rightarrow \langle \downarrow, \rho' \rangle)\}
\]

CP, Case \[
Q(\theta) \equiv \text{FB}_{\text{case}}(M[\overline{\theta}] \rightarrow P_m(\theta), \end{align*}
\]

\[
\{(\rho' \neq 0 : \langle Q(\theta'), \rho \rangle \rightarrow \langle \downarrow, \rho' \rangle)\}
\]

as desired. Note that the last step (**) is due to the behavior of the superoperators in \(\text{FB}_{\text{case}} M[\overline{\theta}] = m \rightarrow P_m(\theta), \text{end}\): when evolved by one transition, they either go to some non-(essentially-aborting)

\[
\langle Q_m, i_m, (\theta') \rangle, M_m, \rho M_m^T \rangle \text{ for some } Q_m, i_m, (\theta) \in \text{Compile}(P_m(\theta)) \text{ for some } m, \text{ or go to } (\text{abort}, M_m, \rho M_m^T) .
\]

Besides, each copy of final state of non-(essentially-aborting)

\[
\langle Q_m, i_m, (\theta') \rangle, M_m, \rho M_m^T \rangle \text{ always terminate in } \langle \theta, 0 \rangle, \text{ therefore not counted on either side of (**).}
\]

4. (While\textsuperscript{T(1)}) By definition, While\textsuperscript{T(2)} reduces to (Case) and (Sequence).

5. (Sum Components) If one of Compile\(P_i(\theta')\) is \{abort\} the statement is immediately true, so we assume otherwise for below. Like in the case case, applying the operational transition rule for 1 step one may observe that (writing "IH" the inductive hypothesis)

\[
0 \neq \rho' \in \{P_i(\theta') + P_i(\theta)\} \rho \quad \text{(C.9)}
\]

IH \[
\begin{align*}
\{\rho' \neq 0 : \langle Q(\theta'), \rho \rangle & \rightarrow \langle \downarrow, \rho' \rangle\} \\
\{\rho' \neq 0 : \langle Q(\theta'), \rho \rangle & \rightarrow \langle \downarrow, \rho' \rangle, Q_m, i_m, (\theta) \in \text{Compile}(P_m(\theta))\}
\end{align*}
\]

CP, Sum Component \[
\begin{align*}
Q(\theta) \equiv \text{Compile}_{\text{case}}(P_i(\theta') + P_i(\theta), \end{align*}
\]

\[
\{(\rho' \neq 0 : \langle Q(\theta'), \rho \rangle \rightarrow \langle \downarrow, \rho' \rangle)\}
\]

as desired! □

D Detailed Proofs from Section 6

Throughout, we use LHS, RHS to denote "left hand side" and "right hand side" resp., and "IH" to denote "Inductive Hypothesis". Wherever applicable, we adopt the overloading convention explained in the main text (See Eqn 6.4).

Before giving proofs, we record the full details for code transformation rule for while\textsuperscript{T(1)} for your interest. Let us denote \(\frac{d}{d\theta}(\text{while}\textsuperscript{T(1)} M[\overline{\theta}] = 1 \text{ do } S(\theta) \text{ done}) \equiv \text{Seq}\textsuperscript{T(1)}\), with:

\[
\text{Seq}\textsuperscript{T(1)} \equiv \text{case } M[\overline{\theta}] = 0 \rightarrow \text{ abort,}
\]

\[
1 \rightarrow \left(\frac{d}{d\theta}(P_i(\theta)); \text{ abort} + (P_i(\theta); \text{ Seq}\textsuperscript{T(1)})\right)
\]

and \(\text{Seq}\textsuperscript{T(2)}\) recursively defined as:

\[
\text{Seq}\textsuperscript{T(2)} \equiv \text{case } M[\overline{\theta}] = 0 \rightarrow \text{ abort,}
\]

\[
1 \rightarrow \left(\frac{d}{d\theta}(P_i(\theta)); \text{ while}\textsuperscript{T(1)} + (P_i(\theta); \text{ Seq}\textsuperscript{T(1)})\right)
\]

Note that Seq\textsuperscript{T(1)} essentially aborts. Let us now introduce some helper lemmas for the soundness proof:

D.1 Technical Lemmas

Lemma D.1. \(U(\theta) \in \{R_\theta(\theta), R_{\varphi \oplus \theta}(\theta)\} : \theta \in \{X, Y, Z\}\) Let \(\frac{d}{d\theta} U(\theta)\) denote the entry-wise derivative of \(U(\theta)\). Then,

\[
\frac{d}{d\theta} U(\theta) &= \frac{1}{2} U(\theta + \pi) \quad \text{(D.1)}
\]

\[
\frac{d}{d\theta} \left(\frac{O}{U(\theta)} \rightarrow U(\theta)\right) = \text{tr}(O \cdot U(\theta) \cdot \rho \cdot \left(\frac{d}{d\theta} U(\theta)\right)') + \text{tr}(O \cdot \left(\frac{d}{d\theta} U(\theta)\right) \cdot \rho \cdot U(\theta)') \quad \text{(D.2)}
\]

\[
\frac{d}{d\theta} \left(\text{tr}(O(U(\theta) \rho U^T(\theta) + U(\theta + \pi) U^T(\theta))) \right) \quad \text{(D.3)}
\]

Proof. Let \(U(\theta) \in \{R_\theta(\theta), R_{\varphi \oplus \theta}(\theta)\} : \theta \in \{X, Y, Z\}\). Then \(U(\theta) = \cos \frac{\theta}{2} I_\theta - i \sin \frac{\theta}{2} \sigma, \) where \(\sigma \in \{X, Y, Z\} \oplus \{X, Y, Z\} \ominus \{Z\}\). In the cases where \(\sigma\) is 1-qubit gate, \(I_\sigma\) denotes identity on that one qubit; likewise for 2-qubit cases. Correctness of Eqns D.1, D.2 basically follows from straightforward computation.

1. Equation D.1:

\[
\frac{d}{d\theta} U(\theta) = \frac{1}{2} \left((-\sin \frac{\theta}{2}) I_\theta - i \cos \frac{\theta}{2} \sigma\right) \quad \text{(D.4)}
\]

\[
= \frac{1}{2} \left(\cos \frac{\theta}{2} + \frac{\pi}{2} I_\theta - i \sin \left(\frac{\theta}{2} + \frac{\pi}{2}\right) \sigma\right) \quad \text{(D.5)}
\]

\[
= \frac{1}{2} \left(\cos \left(\frac{\theta}{2} + \pi\right) I_\theta - i \sin \left(\frac{\theta}{2} + \pi\right) \sigma\right) \quad \text{(D.6)}
\]

\[
= \frac{1}{2} U(\theta + \pi) \quad \text{(D.7)}
\]
2. Equation D.2:
\[
\frac{\partial}{\partial \theta} \left( \langle (O, \rho) \rightarrow U(\theta) \rangle \right) \\
= \frac{\partial}{\partial \theta} \left( tr(U(\theta) \rho U^\dagger(\theta)) \right) \\
= \frac{\partial}{\partial \theta} \left( tr(O \cdot (\cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} \sigma) \cdot \rho \cdot \\
(\cos \frac{\theta}{2} I + i \sin \frac{\theta}{2} \sigma^\dagger) \right) \\
= \frac{\partial}{\partial \theta} \left( tr((\cos^2 \frac{\theta}{2}) \rho \cdot i \cos \frac{\theta}{2} \sin \frac{\theta}{2} \sigma^\dagger) \\
+i \sin \frac{\theta}{2} \cos \frac{\theta}{2} \sigma^\dagger \sigma + \sin^2 \frac{\theta}{2} \sigma^\dagger \sigma^\dagger) \right) \\
= tr(2 \cos \frac{\theta}{2} \cdot (\cos \frac{\theta}{2})' \rho \rho^\dagger) \\
+ tr((\cos \frac{\theta}{2} \sin \frac{\theta}{2})' \rho \rho^\dagger + \cos \frac{\theta}{2} \sin \frac{\theta}{2} \rho \sigma' - \sigma \sigma^\dagger)) \\
+ tr(2 \sin \frac{\theta}{2} \cdot (\sin \frac{\theta}{2} \sigma) \rho \rho^\dagger), \\
\text{as desired.}
\]

On the other hand,
\[
\begin{align*}
tr(O \cdot (U(\theta) \cdot \rho \cdot \frac{d}{d \theta} U(\theta))^\dagger) + \\
tr(O \cdot \frac{d}{d \theta} U(\theta) \cdot \rho \cdot U(\theta)^\dagger)
\end{align*}
\]
\(= \)
\[
\begin{align*}
tr \left( O \cdot (\cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} \sigma) \cdot \rho \cdot \\
(\cos \frac{\theta}{2} I + i \sin \frac{\theta}{2} \sigma^\dagger) \right) + \\
tr(2 \cos \frac{\theta}{2} \cdot (\cos \frac{\theta}{2})' \rho \rho^\dagger) \\
+ tr((\cos \frac{\theta}{2} \sin \frac{\theta}{2})' \rho \rho^\dagger + \cos \frac{\theta}{2} \sin \frac{\theta}{2} \rho \sigma' - \sigma \sigma^\dagger)) \\
+ tr(2 \sin \frac{\theta}{2} \cdot (\sin \frac{\theta}{2} \sigma) \rho \rho^\dagger), \\
\text{as desired.}
\end{align*}
\]

Lemma D.2. Let \(P(\theta) \in q\text{-while}^{(T)}_{\text{raw}}(\theta), P'(\theta) \in q\text{-while}^{(T)}_{\text{raw} \cup \{A\}}(\theta)\). Then for arbitrary \(\theta^* \in \mathbb{R}^k, \rho \in \mathcal{D}(\mathcal{H}_\pi), O \in \mathcal{O}_\pi\),
\[
\begin{align*}
1. \left\langle (O, \rho) \rightarrow P'(\theta^*); P'(\theta^*) \right\rangle & = \left\langle (O, \left\langle P(\theta^*) \right\rangle \rho) \rightarrow P'(\theta^*) \right\rangle, \\
2. \left\langle (O, \rho) \rightarrow P'(\theta^*); P'(\theta^*) \right\rangle & = \left\langle \left\langle P(\theta^*) \right\rangle \right\rangle \left\langle (O, \rho) \rightarrow P'(\theta^*) \right\rangle, \\
\text{with \[\left\langle P(\theta^*) \right\rangle \text{ the dual of } \left\langle P(\theta^*) \right\rangle.}
\end{align*}
\]

\textbf{Proof.} Observe that both \(P(\theta^*); P'(\theta^*)\) and \(P(\theta^*); P'(\theta^*)\) live in \(q\text{-while}^{(T)}_{\text{raw} \cup \{A\}}(\theta)\), so we identify the “smaller” program \(P'(\theta^*)\) with \(I_A \otimes P'(\theta^*), \) where \(I \equiv I_A \circ I_A\) denotes the identity operation on Ancilla.

1. Unfolding definition 5.2,
\[
\begin{align*}
\left( (O, \rho) \rightarrow P(\theta^*); P'(\theta^*) \right) \\
& = \text{tr}(Z_A \otimes \left\langle P(\theta^*) \right\rangle \left\langle I_A \otimes P'(\theta^*) \right\rangle \left((|0\rangle_A \langle 0| \otimes \rho) \right)) \\
& = \text{tr}(Z_A \otimes (\left\langle P(\theta^*) \right\rangle \left\langle (|0\rangle_A \langle 0| \otimes \rho) \right)) \\
& = \left\langle \left( (O, \left\langle P(\theta^*) \right\rangle \rho) \rightarrow P'(\theta^*) \right) \right\rangle, \text{ as desired.}
\end{align*}
\]

2. Observe that:
\[
\begin{align*}
\left( (O, \rho) \rightarrow P'(\theta^*); P'(\theta^*) \right) \\
& = \text{tr}(Z_A \otimes \left\langle P'(\theta^*) \right\rangle \left\langle I_A \otimes P'(\theta^*) \right\rangle \left((|0\rangle_A \langle 0| \otimes \rho) \right)) \\
& = \text{tr}(Z_A \otimes \left\langle P'(\theta^*) \right\rangle \left\langle (|0\rangle_A \langle 0| \otimes \rho) \right)) \\
& = \left\langle \left( (O, \left\langle P'(\theta^*) \right\rangle \rho) \rightarrow P'(\theta^*) \right) \right\rangle.
\end{align*}
\]

\(\text{Fig1b, Sequence}\)
\[
\begin{align*}
\left( (O, \rho) \rightarrow P'(\theta^*); P'(\theta^*) \right) \\
& = \text{tr}(Z_A \otimes \left\langle P'(\theta^*) \right\rangle \left\langle I_A \otimes P'(\theta^*) \right\rangle \left((|0\rangle_A \langle 0| \otimes \rho) \right)) \\
& = \left\langle \left( (O, \left\langle P'(\theta^*) \right\rangle \rho) \rightarrow P'(\theta^*) \right) \right\rangle \left((|0\rangle_A \langle 0| \otimes \rho) \right).
\end{align*}
\]

\(\xi \equiv \left\langle \left( (O, \rho) \rightarrow P'(\theta^*); P'(\theta^*) \right) \right\rangle, (\text{D.18}) \)
\[
\begin{align*}
\left\langle \left( (O, \rho) \rightarrow P'(\theta^*) \right) \right\rangle \\
& = \text{tr}(Z_A \otimes \left\langle P'(\theta^*) \right\rangle \left\langle I_A \otimes P'(\theta^*) \right\rangle \left((|0\rangle_A \langle 0| \otimes \rho) \right)) \\
& = \text{tr}(Z_A \otimes \left\langle P'(\theta^*) \right\rangle \left((|0\rangle_A \langle 0| \otimes \rho) \right)).
\end{align*}
\]

\(\text{D.19}\)

Now RHS (EqnD.18) equals RHS(EqnD.20) via duality, as \(I_A \otimes P(\theta^*)\) does nothing on the ancilla, while \(\left\langle P(\theta^*) \right\rangle \left((O) \in \mathcal{O}_\pi \text{ and } Z_A \text{ are compatible observables.}
\end{proof}

\textbf{Lemma D.3.} Let \(S_m(\theta) \in \text{add-q-while}^{(T)}_{\pi}(\theta)\ (\forall m \in [0, w]), \) where \(w \geq 1\). Denote:
\[
\begin{align*}
\text{Compile}(S_m(\theta)) & \equiv \{Q_{m,0}(\theta), \ldots, Q_{m,s_m}(\theta)\}, \\
\text{Compile}(\sigma(\theta)) & \equiv \{P_{m,0}(\theta), \ldots, P_{m,i_m}(\theta)\}.
\end{align*}
\]

Then for arbitrary \(\rho \in \mathcal{D}(\mathcal{H}_\pi), O \in \mathcal{O}_\pi, \) we have the following computational properties regarding the observable semantics:
\[
\begin{align*}
\left\langle (O, \rho) \rightarrow \frac{\partial}{\partial \theta} (\text{Compile}(S_m(\theta)); S_1(\theta)) \right\rangle & = \\
= \left\langle (O, \rho) \rightarrow \frac{\partial}{\partial \theta} S_2(\theta); S_1(\theta) \right\rangle \\
\end{align*}
\]
\(= \)
\[
= \left\langle (O, \rho) \rightarrow \frac{\partial}{\partial \theta} (\text{Compile}(S_m(\theta)); S_1(\theta)) \right\rangle \\
\end{align*}
\]
\(= \)
\[
= \left\langle (O, \rho) \rightarrow \frac{\partial}{\partial \theta} (\text{Compile}(S_m(\theta)); S_1(\theta)) \right\rangle \\
\end{align*}
\]
\(= \)
\[
= \sum_m \sum_{i_m} \left\langle (O, M_m \rho M_m^\dagger) \rightarrow P_{m,i_m}(\theta) \right\rangle, \\
\text{where } i_m \text{ runs through } [0, t_m].
\]
\(\text{D.24}\)

\(\text{As a reminder:} \) we shall frequently refer to Notations in Equs D.21 and D.22 for the rest of the section.
\[
\begin{align*}
\langle (O, \rho) \rightarrow (\frac{\partial}{\partial \theta} (S_0(\theta) + S_1(\theta)) \rangle & = \langle (O, \rho) \rightarrow \frac{\partial}{\partial \theta} (S_0(\theta)) \rangle + \langle (O, \rho) \rightarrow \frac{\partial}{\partial \theta} (S_1(\theta)) \rangle \\
(\langle (O, \rho) \rightarrow S_0(\theta) + S_1(\theta) \rangle & = \langle (O, \rho) \rightarrow S_0(\theta) \rangle + \langle (O, \rho) \rightarrow S_1(\theta) \rangle.
\end{align*}
\]

(D.25)

(D.26)

(D.27)

(D.28)

Proof: We first make some direct application of the code transformation (hereinafter “CT”, Fig 4) and compilation (hereinafter “CP”, Fig 3) rules; some results will immediately follow from these application.

\[
\begin{align*}
\text{CT,Sequence} & : \text{Compile}(\frac{\partial}{\partial \theta} (S_0(\theta); S_1(\theta))) \\
& = \text{Compile}((\frac{\partial}{\partial \theta} (S_0(\theta)); S_1(\theta))) + \\
& \text{CP.Sum Components} \\
& = \text{Compile}(S_0(\theta); \frac{\partial}{\partial \theta} (S_1(\theta))) \\
& \text{CP.Sequence} \\
& = \text{Compile}(S_0(\theta); \frac{\partial}{\partial \theta} (S_1(\theta))) \quad \text{(D.29)} \\
& \quad \text{Compile}(\frac{\partial}{\partial \theta} (S_0(\theta)); S_1(\theta)) \\
& \quad \begin{cases}
\text{FB}(\text{case } M[\varnothing] = m \rightarrow S_m(\theta) \text{ end}) \\
\text{FB}(\text{case } M[\varnothing] = m \rightarrow S_m(\theta) \text{ end})
\end{cases}
\end{align*}
\]

(D.30)

(D.31)

where \([s_0], [s_1], [t_0], [t_1]\) stand for e.g. \([0, s_0]\), \([0, s_1]\), \([0, t_0]\), \([0, t_1]\) respectively; and

\[
\begin{align*}
\text{CT,Case} & : \text{Compile}(\frac{\partial}{\partial \theta} (\text{case } M[\varnothing] = m \rightarrow S_m(\theta) \text{ end})) \\
& = \text{Compile}(\text{case } M[\varnothing] = m \rightarrow \frac{\partial}{\partial \theta} (S_m(\theta)) \text{ end}) \\
& \text{CP,Case} \\
& = \text{FB}(\text{case } M[\varnothing] = m \rightarrow \frac{\partial}{\partial \theta} (S_m(\theta)) \text{ end});
\end{align*}
\]

also,

\[
\begin{align*}
\text{CT.Sum Component} & : \text{Compile}(\frac{\partial}{\partial \theta} (S_0(\theta) + S_1(\theta))) \\
& = \text{Compile}((\frac{\partial}{\partial \theta} (S_0(\theta)) + \frac{\partial}{\partial \theta} (S_1(\theta))) \\
& \quad \begin{cases}
\text{CP.Sum Component} \\
\text{CP.Sum Component}
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\text{CP.Sum Component} &= \{[P_{0,0}(\theta), \cdots, P_{0,t_0}] \} \\
\text{CP.Sum Component} &= \{[P_{1,0}(\theta), \cdots, P_{1,t_1}] \}.
\end{align*}
\]

(D.32)

(D.33)

(D.34)

(D.35)

As promised, Eqns D.23 and D.26 immediately follows from the results above together with the corresponding definitions. So is Eqn D.27, following the same lines of unfolding by CP-(ND component) laid out in Eqns D.33~D.35 then a direct pattern matching by definition. To obtain D.24, observe that

\[
\begin{align*}
\langle (O, \rho) \rightarrow \frac{\partial}{\partial \theta} (\text{case } M[\varnothing] = m \rightarrow S_m(\theta) \text{ end}) & = \sum_{Q(\theta) \in FB(\text{case } M[\varnothing] = m \rightarrow \frac{\partial}{\partial \theta} (S_m(\theta)) \text{ end})} tr((Z_A \otimes O)(Q(\theta)(((0)_A (0) \otimes m_m M^\dagger_m))) \\
\langle (O, \rho) \rightarrow Q(\theta) & = \sum_{Q(\theta) \in FB(\text{case } M[\varnothing] = m \rightarrow \frac{\partial}{\partial \theta} (S_m(\theta)) \text{ end})} tr(O((Q(\theta))(\rho)) \\
\langle \text{case } M[\varnothing] = m \rightarrow S_m(\theta) \text{ end} & = \sum_{m' \in [0, m_m]} \sum_{j_m \in [0, j_m]} tr((Z_A \otimes O)\text{case } M[\varnothing] = m' \rightarrow m_m M^\dagger_m) \text{ as desired.}
\end{align*}
\]

A few more words on Eqn D.40. Since A is disjoint from \(\mathcal{V} \cong \mathcal{V}\), for each \(Q(\theta) \in FB(\text{case } M[\varnothing] = m \rightarrow \frac{\partial}{\partial \theta} (S_m(\theta)) \text{ end})\), evolving one step per the operational semantics will lead to the configuration \(\langle [P_{m'\cdot j_m}(\theta)]; ((0)_A (0) \otimes m_m M^\dagger_m, \theta)\rangle\), for some non-(essentially-aborting) \(P_{m'\cdot j_m}(\theta)\). Besides, each such configuration appears exactly once affording one summand, due to construction of FB(\(\bullet\)). On the other hand, if \(P_{m'\cdot j_m}(\theta)\) essentially aborts, the trace computed as a summand vanishes, making no contribution to the sum, thereby maintaining the equation.

Eqn D.28 is proved using the same lines of logic:

\[
\begin{align*}
\langle (O, \rho) \rightarrow \text{case } M[\varnothing] = m \rightarrow S_m(\theta) \text{ end} & = \sum_{Q(\theta) \in FB(\text{case } M[\varnothing] = m \rightarrow \frac{\partial}{\partial \theta} (S_m(\theta)) \text{ end})} tr(O((Q(\theta))(\rho)) \\
\langle (O, \rho) \rightarrow Q(\theta) & = \sum_{Q(\theta) \in FB(\text{case } M[\varnothing] = m \rightarrow \frac{\partial}{\partial \theta} (S_m(\theta)) \text{ end})} tr(O((Q(\theta))(\rho)) \\
\langle \text{case } M[\varnothing] = m \rightarrow S_m(\theta) \text{ end} & = \sum_{m' \in [0, m_m]} \sum_{j_m \in [0, j_m]} tr((Z_A \otimes O)\text{case } M[\varnothing] = m' \rightarrow m_m M^\dagger_m) \text{ as desired.}
\end{align*}
\]

\(\square\)
Lemma D.4. Keeping the notations in Lemma D.3, we have
\( \forall \theta^* \in \mathbb{R}^k, O \in O_\tau, \rho \in D(H_\tau), \)
\[
\begin{align*}
\left. \frac{\partial}{\partial \theta} \left( \text{Tr}(O, \rho) \rightarrow S_0(\theta); S_1(\theta) \right) \right|_{\theta^*} & = \sum_{g \in [0,s_1]} \left. \frac{\partial}{\partial \theta} \left( \text{Tr}(O, \left[ \left[ Q_0, g(\theta^*) \right] \right] \rho) \rightarrow S_0(\theta) \right) \right|_{\theta^*} + \sum_{j \in [0,s_1]} \left. \frac{\partial}{\partial \theta} \left( \text{Tr}(O, \left[ \left[ \left[ Q_1, j(\theta^*) \right] \right] \right]^*(O, \rho) \rightarrow S_0(\theta) \right) \right|_{\theta^*} \\
& \quad + \sum_{g \in [0,s_1], j \in [0,s_1]} \left. \frac{\partial}{\partial \theta} \left( \text{Tr}(O, \left[ \left[ \left[ Q_0, g(\theta^*) \right] \right] \right] \rho) \rightarrow S_1(\theta) \right) \right|_{\theta^*} \\
& \quad + \sum_{j \in [0,s_1]} \left. \frac{\partial}{\partial \theta} \left( \text{Tr}(O, \left[ \left[ \left[ \left[ Q_1, j(\theta^*) \right] \right] \right] \right]^*(O, \rho) \rightarrow S_0(\theta) \right) \right|_{\theta^*}
\end{align*}
\]
(D.43)

Proof. Observe:

- \( \text{LHS(D.43)} = \sum_{g \in [0,s_1], j \in [0,s_1]} \left. \frac{\partial}{\partial \theta} \left( \text{Tr}(O, \left[ \left[ Q_0, g(\theta^*) \right] \right] \rho) \rightarrow S_0(\theta) \right) \right|_{\theta^*} + \sum_{j \in [0,s_1]} \left. \frac{\partial}{\partial \theta} \left( \text{Tr}(O, \left[ \left[ \left[ Q_1, j(\theta^*) \right] \right] \right]^*(O, \rho) \rightarrow S_0(\theta) \right) \right|_{\theta^*} \)

- \( \text{RHS(D.45)} = \sum_{g \in [0,s_1], j \in [0,s_1]} \left. \frac{\partial}{\partial \theta} \left( \text{Tr}(O, \left[ \left[ Q_0, g(\theta^*) \right] \right] \rho) \rightarrow S_0(\theta) \right) \right|_{\theta^*} + \sum_{j \in [0,s_1]} \left. \frac{\partial}{\partial \theta} \left( \text{Tr}(O, \left[ \left[ \left[ Q_1, j(\theta^*) \right] \right] \right]^*(O, \rho) \rightarrow S_0(\theta) \right) \right|_{\theta^*} \)

Where the last step is direct application of definition of input-space observable semantics. For the step (***) claiming \( \text{RHS(D.45)} = \text{RHS(D.46)} \), observe that it boils down to the following: for arbitrary \((g^*,j^*) \in [0,s_0] \times [0,s_1],\)
\[
\begin{align*}
\left. \frac{\partial}{\partial \theta} \left( \text{Tr}(O, \rho) \rightarrow Q_{0,g^*}(\theta^*); Q_{1,j^*}(\theta^*) \right) \right|_{\theta^*} & \quad \text{for arbitrary } (g^*,j^*) \in [0,s_0] \times [0,s_1], \\
& = \left. \frac{\partial}{\partial \theta} \left( \text{Tr}(O, \left[ \left[ Q_0, g(\theta^*) \right] \right] \rho) \rightarrow Q_{0,g}(\theta^*) \right) \right|_{\theta^*} + \left. \frac{\partial}{\partial \theta} \left( \text{Tr}(O, \left[ \left[ \left[ Q_1, j(\theta^*) \right] \right] \right]^*(O, \rho) \rightarrow Q_{0,g}(\theta^*) \right) \right|_{\theta^*}
\end{align*}
\]
(D.49)

We argue the correctness of D.49–D.51 below. Simplifying notations, write \( Q_{0,g^*}(\theta^*), Q_{1,j^*}(\theta^*) \) as \( Q_{0}(\theta), Q_{1}(\theta) \) respectively, then let
\[
\begin{align*}
Q_0(\theta) & \equiv \sum_x K_x(\theta) \odot K_x^*(\theta), \\
Q_1(\theta) & \equiv \sum_y J_y(\theta) \odot J^*_y(\theta)
\end{align*}
\]
(D.52)
(D.53)

denote the Kraus operator decomposition of \( Q_0(\theta), Q_1(\theta) \), each with finitely many summands.\(^{11}\)

Then,
\[
\begin{align*}
\left( \frac{\partial}{\partial \theta} \left( \text{Tr}(O, \rho) \rightarrow Q_{g^*}(\theta^*); Q_{j^*}(\theta^*) \right) \right) \bigg|_{\theta^*} & = \frac{\partial}{\partial \theta} \left( \text{tr}(OQ_{0}(\theta)); Q_{1}(\theta) \right) \bigg|_{\theta^*} \quad \text{for arbitrary } (g^*,j^*) \in [0,s_0] \times [0,s_1], \\
& = \frac{\partial}{\partial \theta} \left( \text{tr}(O \sum_y J_y(\theta)(\sum_x K_x(\theta) \rho K_x^*(\theta))) J_y^*(\theta) \right) \bigg|_{\theta^*} \\
& = \sum_{x,y} \left. \frac{\partial}{\partial \theta} \left( \text{tr}(O \sum_x K_x(\theta) \rho K_x^*(\theta)) J_y^*(\theta) \right) \right|_{\theta^*} \\
& = \sum_{x,y} \left. \frac{\partial}{\partial \theta} \left( \text{tr}(O \sum_x K_x(\theta) \rho K_x^*(\theta)) \frac{d}{d \theta} J_y^*(\theta) \right) \right|_{\theta^*} \\
& = \sum_{x,y} \left. \frac{\partial}{\partial \theta} \left( \text{tr}(O \sum_x K_x(\theta) \rho K_x^*(\theta)) \frac{d}{d \theta} J_y^*(\theta) \right) \right|_{\theta^*} \\
& + \sum_{y} \left. \frac{\partial}{\partial \theta} \left( \text{tr}(O \sum_{x} K_x(\theta) \rho K_x^*(\theta)) J_y^*(\theta) \right) \right|_{\theta^*} \\
& + \sum_{y} \left. \frac{\partial}{\partial \theta} \left( \text{tr}(O \sum_{x} K_x(\theta) \rho K_x^*(\theta)) J_y^*(\theta) \right) \right|_{\theta^*}
\end{align*}
\]

\(^{11}\)This is doable because for each \( \theta^* \in \mathbb{R}^k \) we have
\[
\begin{align*}
Q_{0}(\theta^*) & = \sum_{x} K_x(\theta^*) \odot K_x^*(\theta^*), \\
Q_{1}(\theta^*) & = \sum_{y} J_y(\theta^*) \odot J_y^*(\theta^*)
\end{align*}
\]
(D.54)
(D.55)

Since parameterized units have a parameterized-matrix representation, the superoperator \( \text{U}(\theta) = U(\theta) \odot U^\dagger(\theta) \) for any parameterized unitary also have a parameterized operator representation. In our parameterized quantum program syntax only unitaries are parameterized, thus the Kraus operators \( K_x(\theta), J_y(\theta) \)'s can be chosen uniformly. If one insists one may perform a standard structural induction to show the existence of such a parameterized Kraus operator decomposition for any \( Q(\theta) \in \mathbb{R}^k \).

For example, assuming each \( S_{m,j}(\theta) \) decomposes to \( \sum_{l=1}^{r} K_{m,j}(\theta) \odot K_{m,j}(\theta)^\dagger \), then \( M[\theta] = m \rightarrow S_{m,j}(\theta) \) decomposes into
\[
\begin{align*}
\sum_{m} \left( \sum_{l=1}^{r} K_{m,j}(\theta) \odot K_{m,j}(\theta)^\dagger \right) \odot E_m
\end{align*}
\]
As in the unitary case, the linear Kraus operators are entry-wise smooth, ensured by the entry-wise smoothness of matrix representations of parameterized unitaries.
\[
+ \text{tr}(O_j(\theta^*) (\sum_x K_x(\theta^*) \rho \frac{d}{d\theta_j} K_x(\theta^ *)) J_y(\theta^*))) \bigg| \theta^* \bigg) \\
= (b) \frac{\partial}{\partial \theta_j} \text{tr}(O \|Q_1(\theta)\| \|Q_0(\theta)\| \rho) \bigg| \theta^* \bigg) \tag{D.60} \\
+ \frac{\partial}{\partial \theta_j} \text{tr}(O \|Q_1(\theta)\| \|Q_0(\theta)\| \rho) \bigg| \theta^* \bigg) \tag{D.61} \\
\text{S-H Dual} \equiv \left( \frac{\partial}{\partial \theta_j} (\|O \|Q_0(\theta)\| \rho \rightarrow Q_1(\theta)) \right) \bigg| \theta^* \bigg) \tag{D.62} \\
+ \left( \frac{\partial}{\partial \theta_j} (\|Q_1(\theta)\| \|O \| \rho \rightarrow Q_0(\theta)) \right) \bigg| \theta^* \bigg) \tag{D.63} \\
\]

where \( \frac{d}{d\theta} \cdot \) again denotes the entry-wise derivative, and (a), (b) is obtained by unfolding the definition of trace, noticing that all entries of parameterized unitaries are smooth. S-H dual here is short for Schrödinger-Heisenberg dual, as is everywhere else in this section.

\[\square\]

**D.2 Proof Details of Theorem 6.2: Soundness of Differentiation Logic**

**Proof.** As stated in the main text, throughout we fix \( O_A \equiv Z_A \). Unfolding by definition, it suffices to show

\[
\forall O \in O_{\alpha}, \rho \in D(H_{\alpha}), \|O \| Z_A \|, \rho \rightarrow \frac{\partial}{\partial \theta} (S(\theta)) \] 
\[= \frac{\partial}{\partial \theta} (\|O \| \theta \rightarrow S(\theta)) \] 
\[\text{(D.64)}\]

Let \( O \in O_{\alpha}, \rho \in D(H_{\alpha}) \) be arbitrary. We consider each rule in Figure 5 possibly used as the final step of the derivation.

1. (Abort)-(Trivial Unitary): for \( S(\theta) \equiv \text{abort}[\mathbb{V}], \text{skip}[\mathbb{V}] \) or \( q := [0] \), \( \|O \| \theta \rightarrow S(\theta) \| \) is a constant, so \( \frac{\partial}{\partial \theta} (\|O \| \theta \rightarrow S(\theta)) \) is zero. On the other hand, \( \frac{\partial}{\partial \theta} (S(\theta)) \equiv \text{CT,Trivialize} \equiv \text{abort}[U \cup \{A\}], \) meaning \( \|O \| \theta \rightarrow \frac{\partial}{\partial \theta} (S(\theta)) \| = \text{tr}(\|Z_A \| \theta \rightarrow 0) = 0 \).

In Trivial-Unitary, \( \|O \| \theta \rightarrow U(\theta) \| \) doesn’t depend on \( \theta \); since \( \theta \notin \theta \), hence \( \frac{\partial}{\partial \theta} (\|O \| \theta \rightarrow U(\theta)) \| = 0 \), as faithfully represented by the code transformation rule.

2. (Unitary): Assume \( S(\theta) \equiv U(\theta) \) with \( U(\theta) \in \{R_\sigma(\theta), R_{\sigma \otimes \sigma}(\theta)\}_{\sigma \in \{X,Y,Z\}} \). Let \( \sigma \) range over \( \{\sigma, \sigma \otimes \sigma\}_{\sigma \in \{X,Y,Z\}} \) (proof below works for all cases where \( \sigma^2 = I \)). Let \( \frac{d}{d\theta} U(\theta) \) denote the entry-wise derivative of \( U(\theta) \), then recall from Lemma D.1 that

\[
\frac{d}{d\theta} U(\theta) = \frac{1}{2} U(\theta + \pi); \text{ and} \tag{D.65}
\]
\[
\frac{\partial}{\partial \theta} (\|O \| \theta \rightarrow U(\theta)) \| = \text{tr}(O \cdot U(\theta) \cdot \rho \cdot \frac{d}{d\theta} U(\theta) \|) \tag{D.66}
\]
\[
+ \text{tr}(O \cdot \frac{d}{d\theta} U(\theta) \cdot \rho \cdot U^\dagger(\theta)) \tag{D.67}
\]

Let us unfold the definition of \( R_\sigma(\theta) \): 

\[
\text{RHS(D.67)} \equiv \frac{1}{2} \sum_{x,y \in \{0,1\}} \text{tr}(\|Z_A \otimes O\| \|C_R_\sigma(\theta); H_{A} \| \cdot ((|x\rangle A(|y\rangle \otimes \rho)) \\
\|\{0\}_A(|0\rangle \otimes U(\theta) \rho U^\dagger(\theta) \\
+ [0]_A(|1\rangle \otimes U(\theta) \rho U^\dagger(\theta + \pi)) \tag{D.68}
\]
\[
+ [1]_A(|0\rangle \otimes U(\theta + \pi) \rho U^\dagger(\theta)) \tag{D.69}
\]
\[
+ [1]_A(|1\rangle \otimes U(\theta + \pi) \rho U^\dagger(\theta + \pi)) \bigg) \tag{D.70}
\]

\[
\frac{1}{2} \text{tr}(\|Z_A \otimes O\| \left( \sum_{x,y} |x\rangle_A(|y\rangle / 2 \otimes U(\theta) \rho U^\dagger(\theta) \right) \right) \tag{D.71}
\]
\[
+ \sum_x \{ \sum_y |x\rangle_A(|y\rangle \otimes |0\rangle_A(1) / 2 \} U(\theta + \pi) \rho U^\dagger(\theta + \pi) \right) \tag{D.72}
\]
\[
+ \sum_x \{ \sum_y |x\rangle_A(|y\rangle \otimes (|1\rangle_A(0) + |0\rangle_A(1)) / 2 \} U(\theta + \pi) \rho U^\dagger(\theta + \pi) \bigg) \tag{D.73}
\]
\[
U(\theta + \pi) \rho U^\dagger(\theta + \pi) \bigg) \tag{D.74}
\]

Let’s regroup the terms of D.71 ~ D.74, bearing in mind that when observing \( Z_A \otimes O \) on the final state,
the “off-diagonal” part is killed by $Z_A$\footnote{Namely, the trace computation can be decomposed into a finite linear combination of the form $\text{tr}(Z_A \otimes O)|b\rangle_A |b'\rangle_A \otimes |\star\rangle)$ ($b, b' \in \{0, 1\}$), and this term vanishes whenever $b \neq b'$.}. Hence,

\[
\text{RHS}(D.71 \sim D.74) = \frac{1}{4} \text{tr} \left( (Z_A \otimes O) \left( \left| 1 \right\rangle_A \left\langle 1 \right| + |0\rangle_A |0\rangle \right) \right) \tag{D.75}
\]

\[
= \frac{1}{4} \text{tr} \left( U(\theta)\rho U\dagger(\theta) + U(\theta + \pi)\rho U\dagger(\theta + \pi) \right) \tag{D.76}
\]

\[
+ \left| 0 \right\rangle_A \left\langle 0 \right| - \left| 1 \right\rangle_A \left\langle 1 \right| \right) \tag{D.77}
\]

\[
\otimes \left( U(\theta)\rho U\dagger(\theta + \pi) + U(\theta + \pi)\rho U\dagger(\theta) \right) \right) \tag{D.79}
\]

Lastly, the behavior of $Z_A \otimes O$ ensures

\[
\text{RHS}(D.76 \sim D.79) = \frac{1}{2} \text{tr} \left( O(U(\theta)\rho U\dagger(\theta + \pi) + U(\theta + \pi)\rho U\dagger(\theta)) \right) \tag{D.79}
\]

as desired. From this point, we again adopt the notations from Lemma D.3 Eqns. D.21, D.22.

3. (Sum Component)

\[
\text{Lem. D.3} \Rightarrow \frac{\partial}{\partial \theta} \left( \text{RHS}(D.70) \right) \tag{D.80}
\]

\[
\text{IH} \Rightarrow \frac{\partial}{\partial \theta} \left( \text{RHS}(D.71) \right) \tag{D.81}
\]

\[
\text{Lem. D.3} \Rightarrow \frac{\partial}{\partial \theta} \left( \text{RHS}(D.72) \right) \tag{D.82}
\]

\[
\text{Lem. D.3} \Rightarrow \frac{\partial}{\partial \theta} \left( \text{RHS}(D.73) \right) \tag{D.83}
\]

as desired.

4. (Sequence) Assume $S(\theta) \equiv S_0(\theta); S_1(\theta)$; It suffices to show that $\forall \theta^*$,

\[
\left( \frac{\partial}{\partial \theta} \left( \text{RHS}(D.74) \right) \right)_{\theta^*} \tag{D.85}
\]

Let us first manipulate the equation using some computational properties developed from the helper lemmas:

\[
\text{LHS}(D.85) = \frac{\partial}{\partial \theta} \left( \text{RHS}(D.84) \right) \tag{D.86}
\]

\[
\text{Lem. D.3, Eqn. D.23} \Rightarrow \frac{\partial}{\partial \theta} \left( \text{RHS}(D.83) \right) \tag{D.87}
\]

\[
\text{Lem. D.3, Eqn. D.20} \Rightarrow \frac{\partial}{\partial \theta} \left( \text{RHS}(D.82) \right) \tag{D.88}
\]

\[
\text{Lem. D.3, Eqn. D.19} \Rightarrow \frac{\partial}{\partial \theta} \left( \text{RHS}(D.81) \right) \tag{D.89}
\]

This equals, by Lem D.3 Eqns D.30, D.31,

\[
\sum_{\theta \in [0, \pi)} \| (O, \rho) \rightarrow Q_{0,\rho}(\theta^*) ; \rho \| \rightarrow P_{1,\rho}(\theta^*) \| \tag{D.87}
\]

Via Lem D.2, this translates to

\[
\sum_{\theta \in [0, \pi)} \| (O, \rho) \rightarrow Q_{0,\rho}(\theta^*) ; \rho \| \rightarrow P_{1,\rho}(\theta^*) \| \tag{D.88}
\]

\[
\sum_{\theta \in [0, \pi)} \| (O, \rho) \rightarrow Q_{0,\rho}(\theta^*) ; \rho \| \rightarrow P_{1,\rho}(\theta^*) \| \tag{D.89}
\]

where the last step comes from definition of observable semantics with ancilla. Now apply the inductive hypothesis on $S(\theta)$, a universal statement for all $(\rho', \theta') \in \mathcal{D}(\mathcal{H}_T) \times \tilde{O}_T$, to the instances $(\|Q_{0,\rho}(\theta^*) ; \rho\|)$ $(\forall \theta \in [0, \pi))$, we have the first summand

\[
\text{RHS}(D.90) = \sum_{\theta \in [0, \pi)} \left( \frac{\partial}{\partial \theta} \| (O, \rho) \rightarrow S_0(\theta) \| \right) \tag{D.90}
\]

Likewise, apply the inductive hypothesis on $S_0(\theta)$ to the instances $(\rho, \|Q_{1,\theta}(\theta^*) ; \theta\|)$ $(\forall \theta \in [0, s_1))$, we have the second summand

\[
\text{RHS}(D.91) = \sum_{\theta \in [0, \pi)} \| (O, \rho) \rightarrow S_0(\theta) \| \tag{D.91}
\]

Collecting everything, using again a technical lemma from above,

\[
\text{RHS}(D.90) + \text{RHS}(D.91) = \text{RHS}(D.92) + \text{RHS}(D.93) \tag{D.94}
\]

\[
\text{RHS}(D.85), \tag{D.95}
\]

5. (Case) This follows again from the computational lemmas above, and applications of all inductive hypothesis. For conciseness, let us denote by $\text{IH}(m) \rightarrow (M_m \rho M_m^\dagger, O)$ the application of the $m$-th $(m \in [0, w])$ inductive hypothesis to the instance $(M_m \rho M_m^\dagger, O) \in \mathcal{D}(\mathcal{H}_T) \times \tilde{O}_T$.

Then,

\[
\text{Lem. D.4} \Rightarrow \text{RHS}(D.85), \tag{D.96}
\]

The case $M[q] = m \rightarrow S[m](\theta)$ end]}

\[
= \sum_{m} \sum_{i \in [0, t_m]} \| (O, M_m \rho M_m^\dagger) \rightarrow P_{m, i, \rho}(\theta) \| \tag{D.96}
\]

\[
= \sum_{m} \| (O, M_m \rho M_m^\dagger) \rightarrow \| \tag{D.97}
\]
Applying the inductive hypothesis IH(m) → (M_mρM_m^T, O), ∀m, we get that the above equals

\[ \sum_m \frac{\partial}{\partial \theta} (\| (O_m, (M_mρM_m^T_m)) \rightarrow S_m(\theta) \|) = \frac{\partial}{\partial \theta} (\sum_m \| (O, E_mρ) \rightarrow Q_m(\theta) \|) \]

Lem. 3: \[ \frac{\partial}{\partial \theta} (\| (O, ρ) \rightarrow \text{case } M[\theta^*] = m \rightarrow S_m(\theta) \|) = \text{as desired.} \]

6. (While(T)) This is verified by successive application of (Case) rule and (Sequence) rule T times.

□

E Detailed Proofs from Section 7

E.1 Proof for Proposition 7.2: Upper Bound of \( |\# \frac{\partial}{\partial \theta_j} (P(\theta))| \)

Proof. 1. If \( P(\theta) \equiv \text{abort} [\tau] \text{skip} [\tau] | q := | 0 \rangle (q \in \tau) \), then \( \text{OC}_j (P(\theta)) = 0 = |\# \frac{\partial}{\partial \theta_j} (P(\theta))| \) by definition. Similarly, if \( P(\theta) \equiv U(\theta) \) trivially used \( \theta \), no non-aborting programs exists in Compile(\( \frac{\partial}{\partial \theta_j} (P(\theta)) \)), yielding \( |\# \frac{\partial}{\partial \theta_j} (P(\theta))| = 0 \); otherwise, \( |\# \frac{\partial}{\partial \theta_j} (P(\theta))| \) consists of a single \( R^\ast \theta_j \), so both numbers are 1.

2. Assume \( |\# \frac{\partial}{\partial \theta_j} (P_b(\theta))| \leq \text{OC}_j (P_b(\theta)) \) for each \( b \in \{1, 2\} \) and \( P(\theta) \equiv P_1(\theta); P_2(\theta) \). Then (denoting as above Inductive Hypothesis as "IH"):

\[ |\# \frac{\partial}{\partial \theta_j} (P(\theta))| \]

\[ = |\# (P_1(\theta); \frac{\partial}{\partial \theta_j} (P_2(\theta)))| \]

\[ + |\# (\frac{\partial}{\partial \theta_j} (P_1(\theta)); P_2(\theta))| \]

\[ \leq |\# (\frac{\partial}{\partial \theta_j} (P_2(\theta)))| \]

\[ + |\# (\frac{\partial}{\partial \theta_j} (P_1(\theta)))| \]

\[ \overset{\text{IH}}{\leq} \text{OC}_j (P_1(\theta)) + \text{OC}_j (P_2(\theta)) \]

\[ = \text{OC}_j (P(\theta)). \]

where step (⋆) is via rules CT,CP-(ND-Component).

3. Assume \( |\# \frac{\partial}{\partial \theta_j} (P_m(\theta))| \leq \text{OC}_j (P_m(\theta)) \) for each \( m \in [0, \omega] \) and \( P(\theta) \equiv \text{case } M[\theta^*] = m \rightarrow P_m(\theta) \). Then by the CT and CP rules of case, \( \frac{\partial}{\partial \theta_j} (P(\theta)) \) compiles to \( \text{max}_m |\# \frac{\partial}{\partial \theta_j} (P_m(\theta))| \) programs. Assume \( m^* \in [0, \omega] \) is

\[ \text{s.t. } |\# \frac{\partial}{\partial \theta_j} (P_m^*(\theta))| \] attains that maximum. Then,

\[ |\# \frac{\partial}{\partial \theta_j} (P(\theta))| \]

\[ \overset{\text{IH}}{\leq} \text{OC}_m (P_m^*(\theta)) \]

\[ \leq \text{max} \text{OC}_m (P_m(\theta)) \]

\[ = \text{OC}_j (P(\theta)) \]

as desired.

4. If \( P(\theta) \equiv \text{while} (T) M[\theta] = 1 \) do \( P_1(\theta) \) done, it suffices to show \( |\# \frac{\partial}{\partial \theta_j} (P(\theta))| \leq T \cdot |\# \frac{\partial}{\partial \theta_j} (P(\theta))| \). This is true for \( T = 1 \) as \( \frac{\partial}{\partial \theta_j} (\text{while} (1)) \) essentially aborts. By induction if this is true for \( T \), then for \( T + 1 \) we have

\[ |\# \frac{\partial}{\partial \theta_j} (\text{while} (T + 1) M[\theta] = 1 \) do \( P_1(\theta) \) done) | \]

\[ \leq (T + 1) |\# \frac{\partial}{\partial \theta_j} (P_1(\theta))| \]

\[ \overset{\text{Eqn. 2.2}}{\leq} |\# \frac{\partial}{\partial \theta_j} (P_1(\theta); \text{while} (T) M[\theta] = 1 \) do \( P_1(\theta) \) done) | \]

\[ \leq (T + 1) |\# \frac{\partial}{\partial \theta_j} (P_1(\theta))|, \text{ as desired.} \]

F Evaluation Details

F.1 Training VQC instances with controls

For any parameter \( \alpha \), we exhibit the final results of Compile(\( \frac{\partial}{\partial \alpha} (P_1(\Theta, \Phi)) \) and Compile(\( \frac{\partial}{\partial \alpha} (P_2(\Theta, \Phi, \Psi)) \), with \( P_1(\Theta, \Phi) \) and \( P_2(\Theta, \Phi, \Psi) \) defined in the main text:

1. If \( \alpha \in \Theta \): without loss of generality assume \( \alpha = \theta_1 \) (other situations are completely analogous). We denote:

\[ Q'(\Theta) \equiv R'_{X}(\theta_1)|A, q_1); \cdots ; R_Z(\theta_{12})|q_{12} \]


then

\[ \text{Compile}(\frac{\partial}{\partial \alpha} (P_1(\Theta, \Phi)) \equiv \{ | Q'(\Theta); Q(\Phi) \} \]

\[ \text{Compile}(\frac{\partial}{\partial \alpha} (\Theta, \Phi, \Psi)) \equiv \{ | Q'(\Phi) \}

\[ \text{case } M[q_1] = 0 \rightarrow Q(\Phi) \]

\[ 1 \rightarrow Q(\Psi) \} \]

2. If \( \alpha \in \Phi \): without loss of generality assume \( \alpha = \phi_1 \) (other situations are completely analogous). We denote:

\[ Q'(\Phi) \equiv R'_{X}(\phi_1)|A, q_1) ; \cdots ; R_Z(\phi_{12})|q_{12} \]
very similar, and these algorithms mainly differ from each other in their basic "rotate-entangle" blocks using different parameter-qubit combinations. Let $B$ denote the basic rotate-entangle block for QNN which consists of a rotation stage and an entanglement stage – we consider these two stages together a single layer: in the rotation stage, one performs parameterized $Z$ rotations followed by parameterized $X$ rotations and then again parameterized $Z$ rotations on the first 4 (small scale) or 6 qubits (medium or large scale); in the entanglement stage, one performs parameterized $X \otimes X$ rotation on all pairs from the first 4 or 6 qubits. See Figure 7.

Basic block for VQE consists of three stages: the first stage is parameterized $X$ followed by parameterized $Z$; the second stage uses $H$ and CNOT to entangle; the third stage performs parameterized $Z$, $X$, $Z$ in that order. Basic block for QAOA on the other hand, entangles $H$ and CNOT in the first stage, and then performs parameterized $X$ rotations on the second stage.

The more interesting part lies in building multiple layers using control flow, which we will explain using QNN. The small scale if-controlled $QNN_{S,i}$ is two-layered. Let $B$ denote the basic rotate-entangle block for QNN explained as above; $B'$, $B''$ be two similarly-structured rotate-entangle blocks using different parameter-qubit combinations. $QNN_{S,i}$ performs $B$ as the first layer, then measures on the first qubit, and performs $B'$ or $B''$ as the second layer dependent on the measurement output. For medium and large scale if-controlled QNN (i.e. $QNN_{M,i}$ and $QNN_{L,i}$)

| $P(\Theta)$ | OC$(\cdot)$ | $\frac{\# P}{\# q}$ | #gates | #lines | #layers | #qubits |
|------------|-----------|-----------------|--------|--------|--------|----------|
| QNN$_{S,b}$ | 1 | 1 | 20 | 24 | 1 | 4 |
| QNN$_{S,s}$ | 5 | 5 | 20 | 24 | 1 | 4 |
| QNN$_{S,i}$ | 10 | 10 | 60 | 67 | 2 | 4 |
| QNN$_{S,w}$ | 15 | 10 | 60 | 66 | 3 | 4 |
| QNN$_{M,i}$ | 24 | 24 | 165 | 189 | 3 | 18 |
| QNN$_{M,w}$ | 56 | 24 | 231 | 121 | 5 | 18 |
| QNN$_{L,i}$ | 48 | 48 | 363 | 414 | 6 | 36 |
| QNN$_{L,w}$ | 504 | 48 | 2079 | 244 | 33 | 36 |
| VQE$_{S,b}$ | 1 | 1 | 14 | 16 | 1 | 2 |
| VQE$_{S,s}$ | 2 | 2 | 14 | 16 | 1 | 2 |
| VQE$_{S,i}$ | 4 | 4 | 28 | 38 | 2 | 2 |
| VQE$_{S,w}$ | 6 | 4 | 42 | 32 | 3 | 2 |
| VQE$_{M,i}$ | 15 | 15 | 224 | 241 | 3 | 12 |
| VQE$_{M,w}$ | 35 | 15 | 224 | 112 | 5 | 12 |
| VQE$_{L,i}$ | 40 | 40 | 576 | 628 | 5 | 40 |
| VQE$_{L,w}$ | 248 | 40 | 1984 | 368 | 17 | 40 |
| QAOA$_{S,b}$ | 1 | 1 | 12 | 15 | 1 | 3 |
| QAOA$_{S,s}$ | 3 | 3 | 12 | 15 | 1 | 3 |
| QAOA$_{S,i}$ | 6 | 6 | 36 | 41 | 2 | 3 |
| QAOA$_{S,w}$ | 9 | 6 | 36 | 29 | 3 | 3 |
| QAOA$_{M,i}$ | 18 | 18 | 120 | 142 | 3 | 18 |
| QAOA$_{M,w}$ | 42 | 18 | 168 | 94 | 5 | 18 |
| QAOA$_{L,i}$ | 36 | 36 | 264 | 315 | 6 | 36 |
| QAOA$_{L,w}$ | 378 | 36 | 1512 | 190 | 33 | 36 |

Table 3. Output on Test Examples. Note that: (1) $\{S, M, L\}$ stands for "small, medium, large"; $\{b, s, i, w\}$ stands for "basic, shared, if, while"; #lines is the number of lines for input programs. (2) gate count and layer count for $T$-bounded while loop body with $T$. (3) for $\times w$ we have $|\frac{\Theta}{\pi}(P(\Theta))| \leq \text{OC}(P(\Theta))$ because differentiating the unrolled bounded while generates essentially aborting programs which are optimized out of the final multiset.
we have larger rotate-entangle blocks, various parameter-qubit combinations (hence more of the $B'$, $B''$'s involving different parameter-qubit combinations), and more layers of measurement-based control.

(Bounded) while-controlled QNN works similarly: take QNN$_{S,w}$ as an example, it performs the rotate-entangle block $B$ as the first layer, then measure the first qubit; if it outputs 0 we halt; otherwise we perform some $B'$, measure qubit $q_1$ again, halts if outputs 0, performs $B'$ the third time and aborts otherwise.

Note that this is just a verbose way of saying "we wrapped $B'$ with a 2-bounded while-loop"! And similarly, we build more layers on larger systems (QNN$_{M,w}$, QNN$_{L,w}$) using larger rotate-entangle blocks and more layers of bounded-while loops. One should note how bounded while is a succinct way to represent the circuits: as shown in Table 3, in QNN$_{L,w}$ we managed to represent 2079 unitary gates in just 244 lines of code.

We auto-differentiated the three families of quantum programs using our code transformation (hereinafter "CT") and code compilation (hereinafter "CP") rules. As shown in Table 3, the computation outputs the desired multi-set of derivative programs, and the number of non-aborting programs ($|\# \frac{\partial}{\partial \theta}(P(\theta))|$) agrees with the upper bound described in Proposition 7.2. It should be noted that Table 3 indicates our auto-differentiation scheme works well for variously sized input programs, be the size measured by code length, gate count, layer count or qubit count, to name a few.