ABSTRACT

Title of Dissertation: THEORETICAL AND PRACTICAL HIGH-ASSURANCE SOFTWARE TOOLS FOR QUANTUM APPLICATIONS

Yuxiang Peng
Doctor of Philosophy, 2024

Dissertation Directed by: Professor Xiaodi Wu
Department of Computer Science

Quantum computing promises to revolutionize solutions to valuable computational problems like factorization and quantum system simulation. Harnessing the power of quantum computers in real life cannot happen without a supportive software stack. This thesis studies the fundamental problems appearing in the software for quantum computing to shed light on shaping software that controls quantum computing devices in the near term and long future.

First, we consider the software for Hamiltonian-oriented quantum computing (HOQC). We discuss our new software framework, SimuQ, for programming quantum Hamiltonian simulation on analog quantum simulators (AQS). To characterize the capability of AQSs, we introduce new abstractions named abstract analog instruction sets (AAIS), which enable us to design several domain-specific languages and a novel compiler. SimuQ reduces knowledge barriers for the study of HOQC because front-end users can easily program Hamiltonian systems and simulate their dynamics on AQSs. We then demonstrate how SimuQ inspires new algorithm design

and hardware improvements, and extend our view to the full-stack software support of quantum applications via HOQC.

Second, we study the software for circuit-oriented quantum computing (COQC). Many prototypes of quantum programming languages have been proposed in recent years based on quantum circuits, while how to debug programs written in these languages remains a critical challenge. Formal methods, a mature area in programming language research, provide a new perspective in guaranteeing the correctness of quantum programs, and we adapt several formal verification techniques to accommodate the nature of quantum programs. In compilers for quantum while programs, rewrite rules need to preserve the semantics of the programs. We connect non-idempotent Kleene algebra with quantum program semantics and propose an equivalence-proving system with algebraic expression derivation. Besides, deductive program verification can also be leveraged on quantum programs to ensure the semantics correctness. To this end, we formally certify an end-to-end implementation of Shor's algorithm, justifying the feasibility of deductive verification for quantum programs.

THEORETICAL AND PRACTICAL HIGH-ASSURANCE SOFTWARE TOOLS FOR QUANTUM APPLICATIONS

by

Yuxiang Peng

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2024

Advisory Committee:
Professor Xiaodi Wu, Chair/Advisor
Professor Haizhao Yang, Dean's Representative
Professor Andrew M. Childs
Professor Leonidas Lampropoulos
Professor Ming C. Lin

Acknowledgments

Firstly, I would like to express my gratitude to my Ph.D. advisor, Xiaodi Wu. His unique insights have led me to a deeper understanding and exploration of the nature of computing technology development, which eventually oriented us to think of quantum computing from important but underexplored aspects. His mentorship has honed my technical skills and strategic thinking, enabling me to create innovative solutions to critical challenges, and gradually forming my approaches to scientific research. Beyond the scope of research, our extensive collaboration over the last six years has allowed his character to profoundly impact me, making me focus on what truly matters amidst many distractions. These experiences have shaped my life philosophy and will undoubtedly illuminate my path forward.

I want to thank the dissertation committee overseeing this work. Committee chair Xiaodi Wu, dean's representative Haizhao Yang, and members Andrew Childs, Leonidas Lampropoulos, and Ming Lin offered invaluable feedback and diverse perspectives during the dissertation defense. Their insights were instrumental in refining the final version of this dissertation.

The Ph.D. program at the University of Maryland provided me with access to numerous high-quality courses, which generated opportunities to learn from outstanding faculty members and engage with bright classmates. I sincerely thank them because the valuable knowledge gained from these well-structured courses was instrumental in my research and contributed significantly to the composition of this dissertation.

Over the past few years, I have had the privilege of collaborating with numerous talented and amiable researchers. These collaborations have enabled me to carry out research that has culminated in this dissertation. I want to express my gratitude to all these coauthors: Shih-Han Hung, Xin Wang, Shaopeng Zhu, Mingsheng Ying, Haowei Deng, Micheal Hicks, Liyi Li, Finn Voichick, Kesha Hietala, Runzhou Tao, Robert Rand, Jiaqi Leng, Yi-Ling Qiao, Ming Lin, Jacob Young, Pengyu Liu, Joseph Li, Samuel Kushnir, and Lei Fan. Not only have they made significant research contributions, but they have also brought immense joy to my life. I would like to extend my gratitude beyond my direct collaborators to the members of many related organizations. These include Xiaodi Wu's research group, our institute Joint Center for Quantum Information and Computer Science, my research fields (mainly programming language community and quantum computing community), and the University of Maryland. The discussions with these great people have sparked a multitude of exciting research ideas and boundless happiness.

Finally, I want to express my profound gratitude to my family, whose endless support has been crucial in my exploration pursuing academic aspirations. Their continuous guidance and nurturing have shaped the person I am today. I also want to thank my girlfriend, whose continuous support is essential for me to strive through my Ph.D. program.

Table of Contents

List of Tables

List of Figures

Part I

Introduction

Chapter 1: Background

1.1 Brief history of quantum computing

The concept of quantum computing was born out of the difficulty in simulating quantum systems on classical computers. As large-scale electronic computers have become crucial since the last century, the study of large physical systems has depended significantly on classical computers. However, a significant challenge is encountered in the realm of quantum physics: quantum systems necessitate an exponential number of classical bits for state representation, making simulation even more complex. Richard Feynman, in a seminal lecture [1], suggested the emulation of a quantum system through another precisely-controlled quantum system—quantum computers—which inherently bypass the curse of dimensionality. This concept evolved into quantum Hamiltonian simulation, which is a fundamental component of Hamiltonian-oriented quantum computing.

Theoretical computer scientists and mathematicians later started studying the computational power of quantum computers. They introduced new concepts such as quantum bits (qubits), quantum gates, and quantum circuits to mimic the computation units of classical computers. A quantum bit (qubit) comes from selected two-level systems in quantum systems, such as atoms or ions, and it is possible to simulate any n-level quantum system efficiently using $O(\log n)$ qubits. Quantum states are represented by finite-dimensional complex vectors, and quantum processes

unfold in discrete time through quantum circuits composed of numerous quantum gates. When a quantum gate acts on qubits, it alters the quantum state. This abstraction of quantum gates hides the natural continuous-time evolution of quantum states but reveals the initial and final states, aiding theorists in the development of quantum algorithms.

In the early 1990s, several quantum algorithms were introduced using these abstractions and showing demonstrated quantum advantage, such as the Deutsch-Jozsa algorithm. This led to the so-called circuit-oriented quantum computing (COQC) paradigm. The most famous one among these, Shor's algorithm, was proposed in 1997 [2], offering a polynomial-time algorithm to the large integer factorization problem for the first time with the aid of quantum computers. This discovery significantly increased the potential impact of quantum computers, particularly since Shor's algorithm could attack RSA encryption, posing a threat to contemporary cryptographic systems. Concurrently, Grover's algorithm was introduced for searching unstructured databases, showcasing another instance of quantum speedup. Together, Shor's and Grover's algorithms lay the cornerstones for the development of subsequent quantum algorithms.

Algorithm development for quantum computers has advanced rapidly over recent decades, yet the practical realization of these algorithms by modern hardware remains a distant prospect. Various quantum computer architectures have been proposed, utilizing precisely engineered physical devices such as superconducting chips, Rydberg atom arrays, and trapped ion arrays. These systems aim to achieve universal quantum computing through the construction of high-fidelity quantum gates. However, for the time being and the foreseeable future, they function primarily as analog quantum simulators, without the capability to execute thousands of quantum gates. This limitation stems from the substantial overheads associated with algorithm and gate implementation through circuit abstractions in COQC. The general belief is that it may take another 20

years to witness a fault-tolerant quantum computer being used in practical applications through COQC. In the interim, the potential of quantum computers is likely to be exploited via HOQC.

1.2 Quantum software stacks

Quantum computing applications are largely dependent on the software that manipulates computations on the hardware. However, the attention on quantum software research has not kept pace with the focus on hardware and theoretical developments. Since 2000, there have been emerging prototypes of quantum programming languages and compilers, primarily developed through COQC using circuit abstraction.

A stack of quantum software through circuit abstraction consists of multiple layers of software abstractions lying between applications and hardware. From top to down, several major components include:

- software interface for inputting application problems;
- high-level programs for quantum operations (e.g., data structures and algorithms);
- circuit-level programs describing the gate sequences in quantum circuits;
- hardware-level control software describing the pulses.

Then a compilation scheme transforms programs among the layers sequentially, including significant components like:

- implementation of quantum algorithms for specific applications in high-level languages;
- circuit generation from high-level programs;

- circuit optimization to reduce the size;
- device-specific quantum gate implementations using pulses;

On the other side, software tools for HOQC are lacking. The research on the theory of Hamiltonian-based quantum algorithms is relatively rare, leading to lower attention in the software development of HOQC. However, HOQC avoids many overheads compared to COQC, which is crucial for near-term quantum applications. We will discuss more details in Chapter 3.

During the programming of the components in the scheme, a critical challenge appears: how to debug quantum programs? Human errors are inevitable in the construction of the software toolchain, and to have useful results for applications, we need a bug-free software suite. Conventional debugging techniques like assertions and testing fall short because of the different nature of quantum mechanics. Alternatively, a family of mature techniques from programming language research comes to the save of ensuring the correctness of the programs. Formal verification, where people design formal techniques to verify the correctness of programs, can be adapted to quantum programs. The research of formal verification for quantum programs has drawn more and more attention in recent years, because of its unique advantages in the new scenario.

Chapter 2: Dissertation Overview

We study the software tools for both HOQC and COQC in this dissertation. For HOQC, we focus on building the foundational software tools in Part II to serve as a platform for HOQC studies. For COQC, we study formal methods in Part III to ensure the correctness of quantum programs.

Framework for programming Hamiltonian simulation

Currently, there is little study in the literature on programming languages for HOQC. Since Hamiltonian-based quantum algorithms aim for practical cases, they need empirical studies to understand performances. However, manual mapping of designed Hamiltonian to analog quantum simulators' Hamiltonian is tedious and error-prone (for human programmers). A software stack is necessary to support programmers in programming Hamiltonian simulation.

In Chapter 4, we propose SimuQ, a framework for programming and deploying Hamiltonian simulation. With SimuQ, front-end users can easily program many-body Hamiltonian systems with a domain-specific language, Hamiltonian Modeling Language, and compile the program into executable pulses using the SimuQ compiler. Currently, SimuQ supports multiple platforms of backends, including neutral atom arrays, trapped ion arrays, and superconducting circuits. This is achieved by a novel abstraction of analog quantum simulators, namely abstract

analog instruction sets (AAIS), which capture the programmability of these devices. Then we may use instruction schedules to describe the Hamiltonian evolution of devices. Hardware developers can design AAISs for their own devices and program them with another domain-specific language, AAIS Specification Language.

Based on the new abstraction, we can build a compiler to automatically generate the schedules according to the front-end user's Hamiltonian and the device instruction set. We decompose the compilation procedure to multiple compiler passes to reduce the problem into several natural steps, with several intermediate representations. The most significant step in the compiler is to synthesize the target Hamiltonian with descriptions of the abstract analog instruction sets. We model this problem as a mixed integer equation system and leverage a least-square-based solver to obtain solutions.

SimuQ provides a testbed for full-stack quantum technology evaluation. From applications to hardware, we evaluate quantum algorithms, compilation techniques, and hardware techniques with SimuQ to understand how significant the new idea in the stack is.

Differentiable analog quantum computing

With the new abstraction of instruction schedules which we study in SimuQ, new algorithmic ideas emerge in HOQC. Inspired by SimuQ, we study a new Hamiltonian-based algorithm in Chapter 5 to evaluate the gradient for parameterized Hamiltonian systems.

A parameterized Hamiltonian system is controlled by a vector of variables $\mathbf{v}$ and measured by an arbitrary observable M. Our algorithm can estimate $\frac{\partial \langle \psi(T)|M|\psi(T)\rangle}{\partial \mathbf{v}}$ with a hybrid digital-analog quantum algorithm, where $|\psi(T)\rangle$ is a quantum state obtained by evolving from an initial

state $|\psi_0\rangle$ obeying the Schrödinger's equation with the parameterized Hamiltonian. By applying a classical optimizer to update $\mathbf{v}$, we may optimize the pulse control of quantum systems.

The parameterized Hamiltonian evolution can model many quantum applications. For example, quantum control asks for a series of pulses that can prepare a state or realize a quantum gate on a quantum device. These tasks can be formulated as an optimization of form $\langle\psi(T)| M |\psi(T)\rangle$, and be tackled by gradient descent algorithms. Similarly, quantum optimization algorithms like variational ground state estimation and quantum approximate optimization algorithms can also be adapted to our setting with our algorithm.

Deductive verification of Shor's algorithm

A significant technique in formal methods to ensure program correctness is deductive program verification. It is a technique based on the Curry-Howard isomorphism, where the correspondence between logic and type systems makes mechanical proof checking possible with a type checker. Many proof assistants allow programmers to specify mathematical theorems and prove them mechanically. Deductive program verification then adapts this idea to mechanically certify the correctness of programs. Programmers are required to write the target program (or their equivalences) in the proof assistant, together with the specification of the program semantics and a mechanical proof. Then the proof assistant certifies that the semantics of the program satisfy the desired behavior, concluding the correctness of the program.

This technique is adapted for quantum circuit generating programs [3, 4], where formal definitions of complex matrices, quantum circuits, and their denotational semantics (the unitary matrices) are formulated in proof assistants like Coq and Isabella. Many quantum algorithms

like the Deutsch-Jozsa algorithm [3] and Grover's algorithm [4] are implemented in these proof assistants and formally certified. Besides, related quantum software like circuit optimizer with many easy-to-reason rules is also implemented and verified.

It is natural to whether deductive verification can handle most scenarios in quantum software. This question motivates us to look into the most sophisticated quantum algorithm, Shor's algorithm for factorizing big integers. This is a hybrid quantum-classical algorithm whose correctness heavily depends on theorems from number theory. Previous attempts [5] at verifying its correctness focus on the quantum subroutine. We construct an end-to-end implementation of Shor's algorithm and formally certify that the failure probability of our programs is less than a constant with polynomial resources, with details presented in Chapter 7.

Our implementation contains an in-place modular multiplication circuit generator, which is viewed as an oracle in Shor's algorithm. Its implementation typically requires a translation from classical reversible circuits to quantum circuits. To ease the challenges in the implementation and reasoning, we propose a reversible circuit intermediate representation for classical reversible circuit construction. Meanwhile, we prove a theorem transferring the correctness of a classical reversible circuit into the correctness of its corresponding quantum oracle.

The classical parts of Shor's algorithm also rely on many number theory results. The post-processing makes use of the continued fraction expansion algorithm, whose correctness depends on Legendre's theorems. Shor also reduces the factorization problem into order finding problem, whose proof relies on the 2-adic number properties. We formally prove these theorems in Coq and establish the bounds for the failure probability of Shor's algorithm. Besides, we also analyze and prove bounds on the resource consumption of Shor's algorithm including the number of qubits and number of gates.

To validate the correctness of the specification of our implementation, we extract the programs into OCaml which can generate quantum circuits in OpenQASM format. By employing DDSIM, a classical simulator of quantum circuits, we empirically show that the success probability in small-scale cases matches the theoretical bounds we proved in Coq.

Equivalence theory for quantum while programs

Another essential component in the software stack for COQC is the compiler that integrates optimization techniques, where many rewrite rules are realized to reduce the resource consumption of quantum programs. Bugs in the optimizer may be triggered only in rare cases, leading to incorrect rewrite rules hard to detect. We study formal methods for ensuring the correctness of quantum while program's rewrite rules, effectively establishing an equivalence theory of quantum while programs.

Kleene algebra models the equivalence relations of regular languages. Since the control flow of classical deterministic programs shares similar structures of regular expressions, Kleene algebra is employed to algebraically derive equivalences of classical programs [6]. [7, 8, 9] adds tests to Kleene algebra, enabling the analysis of assertions and logic in programs.

In light of the widespread applications of Kleene algebra (with tests) in program analysis, we connect a variant of Kleene algebra, the non-idempotent Kleene algebra (NKA), to quantum programs, illustrated in Chapter 8. We build an isomorphism between NKA and a generalized version of quantum channels and establish a theorem enabling proving quantum program equivalence by algebraic expression derivation. With the theoretical tool, we employ NKA to reason about several compiler rules for quantum while programs and establish the first algebraic proof

of a normalization theorem for quantum while programs.

To further subsume quantum Hoare logic, we embed an effect algebra into NKA to create the non-idempotent Kleene algebra with tests. It enables mechanical derivation of quantum Hoare triples with succinct expressions.

Part II

Build HOQC Software Stacks

Chapter 3: Part II Preface

Hamiltonian-oriented quantum computing is a paradigm of quantum computing where Hamiltonian is employed as the first-principle abstraction of quantum computing. A stack connecting applications to quantum hardware with HOQC needs a refurbishment of intermediate abstraction layers in the current stack of COQC using Hamiltonian as the abstraction:

- **HOQC algorithm design.** For applications, theorists design quantum algorithms with a description in the form of continuous-time Hamiltonian evolution. The major difference between Hamiltonian-based algorithms and circuit-based algorithms is that the components in the HOQC algorithms are compositions of Hamiltonians. Although in principle these algorithms can be implemented with a digital quantum computer, the overheads are still unaffordable for modern architectures. For example, the implementation of the product formula method for simulating a 100-site Heisenberg model needs more than 10^9 quantum gates. There are several existing HOQC algorithms like quantum annealing, conventionally under the category of analog or analog-digital hybrid algorithms.

- **Hamiltonian compilation.** The Hamiltonian designed in HOQC algorithms needs to be mapped to quantum hardware's Hamiltonian to be executed. On a quantum platform, only a few types of Hamiltonian can be natively simulated, hence compilation techniques are required to embed a target Hamiltonian into the device Hamiltonian. The theory of univer-

sal Hamiltonian simulation states that arbitrary k-local Hamiltonian can be embedded into the low-energy subspace of certain fixed types of Hamiltonian with Hamiltonian gadgets. However, since the theory focuses on general cases, the gadget design is still expensive, and universal Hamiltonian is also rarely native to devices. Embedding schemes of specific applications with low overheads for specific devices need fine-grained design and optimization, which should be integrated into the compilation process.

- **Hardware programming.** The eventual device Hamiltonian is implemented through physical pulses as mediums, such as microwaves and lasers interacting with the hardware. A control software generates the pulses, modulates them with physical pulse generators, and sends the pulses to the devices.

The benefit of HOQC compared to COQC is the low overhead in each layer. In COQC, the resource consumption of many algorithms in practical cases requires thousands of logical qubits and millions of logical quantum gates. For example, using Shor's algorithm to factorize 2048-bit RSA integers demands more than 10^4 qubits and 10^7 gates. However, extant quantum computers can implement at most 1000 qubits and several hundreds of quantum gates before the fidelity of the state fidelity deteriorates to unacceptable levels. However, in HOQC, the overheads are considerably lower, because Hamiltonian is the native description for many application problems (like simulation of quantum systems) and also the hardware. By avoiding the detour to quantum circuits and directly mapping target Hamiltonian realizing algorithms to device Hamiltonian, several orders of magnitudes of overheads can be saved.

Because of the low overhead of HOQC, it is more likely to realize practical applications of quantum computing via HOQC. This future was once a reality for classical computing. In

the early history of classical computing, analog computers made use of mechanical or electronic components to build systems with native continuous variables and continuous time evolution, targeting problems in nature like missile trajectory calculation or network analysis. Later, after the blooming development of electronic components, integrated circuits leveraged their great scalability to realize large-scale general-purpose digital computation, which was efficient enough to overcome the overheads. We believe that in the near future, HOQC will be more valuable in applications, while in the far future, COQC will benefit from error correction and easy design automation to surpass HOQC in many applications.

However, the research efforts in building a stack for HOQC are limited in the literature. In this chapter, we study several software tools that help with the programming and control of quantum devices via HOQC. The overarching goal is to build an end-to-end software stack that utilizes Hamiltonian-oriented programming designs to reduce overheads in the stack.

In Chapter 4, We first study a foundational framework for Hamiltonian-oriented programming, named SimuQ. It provides a platform for further empirical exploration of HOQC. This chapter is based on [10]: Peng, Y., Young, J., Liu, P., & Wu, X. (2024). SimuQ: A Framework for Programming Quantum Hamiltonian Simulation with Analog Compilation. *Proceedings of the ACM on Programming Languages, 8*(POPL), 2425-2455.

Follow-up studies are based on SimuQ to investigate technology in multiple layers of the software stack for HOQC. In Chapter 5, Specifically, we study an algorithm for automatic differentiation of parameterized Hamiltonian systems, inspired by SimuQ and designed for the stack of HOQC. This chapter is based on [11]: Leng, J., Peng, Y., Qiao, Y. L., Lin, M., & Wu, X. (2022). Differentiable analog quantum computing for optimization and control. *Advances in Neural Information Processing Systems, 35*, 4707-4721.

Chapter 4: SimuQ: A Framework for Hamiltonian-Oriented Programming

Chapter summary

Quantum Hamiltonian simulation, which simulates the evolution of quantum systems and probes quantum phenomena, is one of the most promising applications of quantum computing. Recent experimental results suggest that Hamiltonian-oriented analog quantum simulation would be advantageous over circuit-oriented digital quantum simulation in the Noisy Intermediate-Scale Quantum (NISQ) machine era. However, programming analog quantum simulators is much more challenging due to the lack of a unified interface between hardware and software.

In this chapter, we design and implement SimuQ, the first framework for quantum Hamiltonian simulation that supports Hamiltonian programming and pulse-level compilation to heterogeneous analog quantum simulators. Specifically, in SimuQ, front-end users specify the target quantum system with Hamiltonian Modeling Language, and the Hamiltonian-level programmability of analog quantum simulators is specified through a new abstraction called the abstract analog instruction set (AAIS) and programmed in AAIS Specification Language by hardware providers. Through a solver-based compilation, SimuQ generates executable pulse schedules for real devices to simulate the evolution of desired quantum systems, which is demonstrated on superconducting (IBM), neutral-atom (QuEra), and trapped-ion (IonQ) quantum devices. Moreover, we demonstrate the advantages of exposing the Hamiltonian-level programmability of devices with native

operations or interaction-based gates and establish a small benchmark of quantum simulation to evaluate SimuQ's compiler with the above analog quantum simulators.

4.1 Introduction

Developing appropriate abstraction is a critical step in designing programming languages that help bridge the domain users and the potentially complicated computing devices. Abstraction is a fundamental factor in the productivity of the underlying programming language. Prominent early examples of such include, e.g., FORTRAN [12] and SIMULA [13], both of which provide high-level abstractions for modeling desirable operations for domain applications and have been proven enormous successes in history.

Conventionally, abstractions for quantum computing adopt (qubit-level) quantum circuits to describe procedures, a mathematically simple approach that works well as a mental tool for the theoretical study of quantum information and algorithms [14, 15]. As a result, many quantum programming languages [3, 16, 17, 18] have adopted quantum circuits as the only abstraction. Many quantum applications are implemented using these programming languages to generate quantum circuits, although only a few can be demonstrated on existing quantum devices.

Quantum Hamiltonian simulation (also called quantum simulation[1]) is arguably one of the most promising quantum applications. The evolution of a quantum system, starting from a quantum state represented by a high-dimensional complex vector $|\psi(0)\rangle$, obeys the *Schrödinger*

[1]In certain contexts, *quantum simulation* and *quantum simulators* refer to the classical simulation of quantum circuits and the corresponding classical software tools, respectively. Yet throughout this chapter, quantum simulation represents the task of simulating a quantum Hamiltonian system, and quantum simulators represent controllable quantum devices that are capable of simulating other quantum systems.

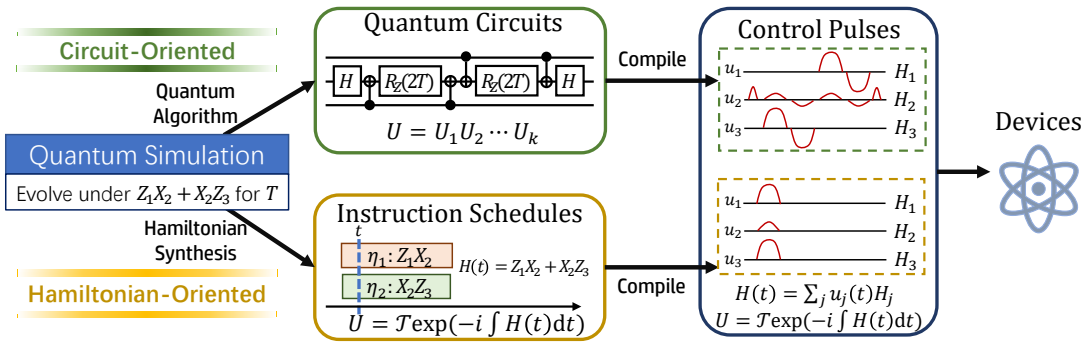


Figure 4.1: The circuit-oriented and Hamiltonian-oriented schemes for compiling quantum Hamiltonian simulation on quantum devices. Here $\mathcal{T}\exp(-i\int H(t)\mathrm{d}t)$ is a solution to a Schrödinger equation governed by $H(t)$.

equation:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left|\psi(t)\right\rangle = -iH(t)\left|\psi(t)\right\rangle, \tag{4.1.0.1}$$

where $H(t)$ is generally a time-dependent Hermitian matrix, also known as the *Hamiltonian* governing the system. Probing quantum phenomena from solutions of the *Schrödinger* equation is a promising approach to tackle many open problems in various domains, including quantum chemistry, high-energy physics, and condensed matter physics [19, 20, 21]. However, for an n qubit system, the dimension of both $H(t)$ and $|\psi(t)\rangle$ could be 2^n, which makes its classical simulation exponentially difficult in general. Though mature software developments for classical simulation of quantum systems using methods like quantum Monte Carlo [22] and density-matrix renormalization groups [23, 24] succeed for restricted cases, many intermediate-size ($\sim$100 sites) quantum systems of significance are still out of reach for classical computers.

To address this issue, in his famous 1981 lecture, Feynman [1] suggested employing a precisely controlled quantum system to simulate a target quantum system to avoid exponential complexity. Modern quantum technologies foster a variety of platforms to advance the realiza-

tion of Feynman's proposal, for example, photonic systems [25], superconducting circuits [26], semiconductor nanocrystals [27], neutral atom arrays [28], and trapped-ion arrays [29]. Most of them are described by a Hamiltonian with continuous-time parameters characterizing the signals sent through controllable physics instruments like microwaves or magnetic fields. They are called *analog quantum simulators*. Only a few devices support a specific set of system evolutions with sophisticated pulse engineering, abstracted as a set of universal quantum gates [30], hence called *digital quantum computers*. They include IBM's superconducting devices [31] and IonQ's trapped-ion devices [32]. However, inherent noises on near-term digital quantum computers induce detrimental errors causing short coherence time (i.e., quantum states do not deteriorate to classical states within it) and preventing demonstrating large quantum applications with provable speedup. The solution through fault-tolerant quantum computing [33] requires significantly lower gate implementation errors and better device connectivity, impractical in the NISQ era [34].

Motivated by the experimental success of simulation by designing and building specific precisely controlled quantum systems mimicking the Hamiltonian of target quantum systems [35, 36, 37, 38], programming analog quantum simulators in a Hamiltonian-oriented scheme is a promising approach to quantum applications before fault-tolerant digital quantum computers are manufactured. Instead of programming quantum circuits implementing quantum simulation algorithms, Hamiltonian-oriented schemes directly program Hamiltonians of analog quantum simulators to synthesize an evolution equivalent to the desired quantum system evolution. Analog quantum simulators have native support for generating Hamiltonians, resulting in a succinct translation process to construct pulse schedules. Via Hamiltonian programming, complicated interactions that demand sophisticated quantum algorithms and large quantum circuits to simu-

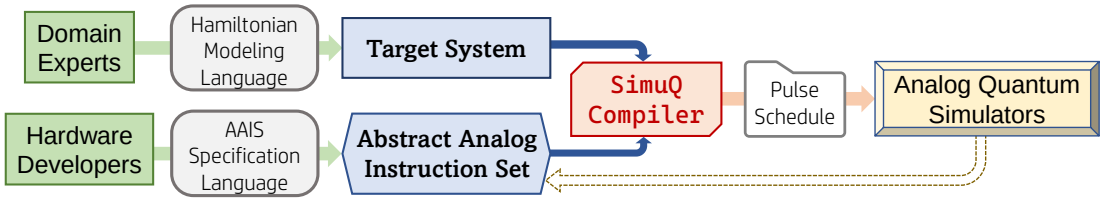


Figure 4.2: The framework of SimuQ is presented. Here abstract analog instruction sets are designed and programmed by hardware developers based on the capability of their analog quantum simulators.

late can be natively constructed and simulated on analog quantum simulators. We compare both schemes for quantum simulation in Figure 4.1 with further details.

By breaking the quantum circuit abstraction and exposing the Hamiltonian-level programmability of modern quantum devices, resource-efficient protocols can deliver reliable solutions to quantum applications [39] on NISQ devices, including various devices that do not support universal quantum gates, like QuEra's neutral atom devices.

Although Hamiltonian-oriented approaches for quantum simulation are beneficial, there is a lack of formal abstractions and supplementary software stacks. Prior works of analog quantum simulation following Hamiltonian-oriented schemes [37, 38] manually construct device-specific configurations, which are tedious, error-prone, and demanding for hardware knowledge, hence not suitable for large-scale experiments.

We propose SimuQ with the first end-to-end automatic framework for quantum simulation on general analog quantum simulators, illustrated in Figure 4.2. As a result, domain experts can focus on describing the desired quantum simulation problems and leave their implementation and deployment to the automation of SimuQ. Our framework lays the foundation for large-scale applications of analog quantum simulators, paving the path for a wide range of novel and practical solutions to domain problems via quantum Hamiltonian simulation for common users.

Physics	Signal carriers	Pulse signals	Signal effects	Device evolution
Rydberg devices	Laser emitters	Time-dependent lasers	$H^{(j)}_{\text{laser}}$	Obeys $H_{\text{Rydberg}}(t)$
AAIS	Signal lines	Instructions	Instruction Hamiltonians	Total Hamiltonian

Table 4.1: Comparison among physics concepts, Rydberg devices instances, and AAIS abstraction designs.

Related Works. There are a few Hamiltonian-level programming interfaces for analog quantum simulators, such as IBM Qiskit Pulse [40], QuEra Bloqade [41], and Pasqal Pulser [42] developed by hardware service providers. These interfaces are designed to represent the specific underlying quantum hardware rather than to provide a unified interface for all analog quantum simulators like AAIS. Computational quantum physics packages like QuTiP [43] support modeling and numerical calculation of quantum simulation without any compilation to quantum devices. Software tools for quantum Hamiltonian simulation are discussed extensively for circuit models [44, 45, 46, 47, 48], while the expressiveness of the circuit abstraction limits their exploitation of analog quantum simulators. SimuQ's solver-based compilation is inspired by the seminal work in classical analog compilation [49, 50]. However, the specific abstraction and compilation technique therein is less relevant as the nature of analog quantum devices is very different from classical ones.

4.2 Abstract Analog Instruction Set

An abstract analog instruction set (AAIS) conveys the functionality of an analog quantum simulator in the form of instructions and system Hamiltonians, including necessary device information for synthesizing target quantum systems.

We present the AAIS design and their physics correspondences in Table 4.1. An *analog instruction* η of an AAIS contains configurable parameters $\vec{v}$ and generates an *instruction Hamil-*

tonian $H_\eta(\vec{v})$ on the device when executed. These parameters are *local variables* of η, modeling the device parameters that can change over time. The instruction Hamiltonian $H_\eta(\vec{v})$ takes the following form where $u_P(\vec{v})$ is a real function depending on the local variables $\vec{v}$:

$$H_\eta(\vec{v}) = \sum\nolimits_P u_P(\vec{v}) \cdot P. \tag{4.2.0.1}$$

Additionally, a *system Hamiltonian* $H_{\text{sys}}(\vec{v}_{\text{glob}})$ with a similar form of (4.2.0.1) applies an always-on effect on the device. A vector $\vec{v}_{\text{glob}}$ of time-independent configurable parameters, called *global variables*, belongs to it. These global variables are configured before executing any instructions and stay unchanged during the execution.

4.3 Domain-specific languages

SimuQ is the first framework to tackle quantum simulation with Hamiltonian-level compilation to analog quantum simulators. It includes a collection of novel abstractions and domain-specific languages (DSL). We propose two DSLs in SimuQ: Hamiltonian Modeling Language (HML) for front-end users to depict their target quantum systems and AAIS Specification Language (AAIS-SL) to specify analog abstract instruction sets (AAISs) of analog quantum simulators.

4.3.1 Hamiltonian Modeling Language

HML is a DSL designed to describe the physical structure of many-body quantum systems that introduces many abstractions, including quantum sites and site-based representations of Hamiltonians. We implement this language in Python, with its abstract syntax and denotational

$$A \in \text{Site}, \qquad r \in \mathbb{R}, \tau \in \mathbb{R}^+$$

$$\begin{aligned}
R \in \text{Operator} \quad &::= I \mid X \mid Y \mid Z \\
S \in \text{Scalar} \quad &::= S_1 + S_2 \mid S_1 \cdot S_2 \mid S_1 - S_2 \mid S_1/S_2 \\
&\quad \mid \exp(S) \mid \cos(S) \mid \sin(S) \mid r \\
M \in \text{Hermitian} \quad &::= M_1 + M_2 \mid M_1 \cdot M_2 \mid S \cdot M \mid A.R \\
E \in \text{Evolution} \quad &::= \textbf{nil} \mid (M, \tau); E
\end{aligned}$$

(a) Abstract syntax of HML.

$$\begin{aligned}
h_{A.R} &= R_A \\
h_{S \cdot M} &= \text{eval}(S) \cdot h_M, \\
h_{M_1 + M_2} &= h_{M_1} + h_{M_2}, \\
h_{M_1 \cdot M_2} &= h_{M_1} \cdot h_{M_2}, \\
[\![\textbf{nil}]\!] &= I, \\
[\![(M, \tau); E]\!] &= [\![E]\!] \cdot e^{-i\tau h_M}.
\end{aligned}$$

(b) Semantics of HML.

Figure 4.3: Syntax and denotational semantics of HML. Here Site contains system sites. h_M translates to the Hermitian matrix described by M. R_A is a Hermitian matrix where operator R applies to site A and I applies to other sites. Function eval evaluates scalar expression S to a real number.

semantics formally defined in Figure 4.3.

Abstract Syntax of HML The first-class objects in HML are *sites of quantum systems*. A site is an abstraction for any quantized 2-level physical entity, like atoms with two energy levels, whose mathematical description is a qubit. In HML, site identifiers are collected in a set Site, each representing a site of the system. Four operators, I, X, Y, and Z, are defined to represent the Pauli operators, and they are *site operators*. We denote the X operator of qubit q as $q.X$ and other operators similarly.

A time-independent Hamiltonian is effectively a Hermitian matrix programmed by algebraic expressions. The basic elements are site operators $A.R$. Expressions for Hermitian are constructed using site operators and scalar expressions, consisting of common matrix operations and scalar operations. An evolution E in HML is a sequence of pairs (M, τ), representing a sequential evolution where each segment is governed by a time-independent Hamiltonian h_M and for time τ.

Remark 4.3.1. *Beyond sites representing qubits, sites representing fermionic and bosonic modes can be defined together with their annihilation and creation operators. These are characterized by different types of sites in our implementation. Each type of site contains specific site operators, and the operator algebras are symbolically implemented. We omit formal discussions of them for simplicity.*

Remark 4.3.2. *HML can generally deal with Hamiltonians with continuous-time coefficients by introducing an additional identifier t in scalars. We choose sequences of time-independent evolution for numerical convenience in the compilation stage and leave this possibility for the future.*

Semantics of HML The denotational semantics of a program E in HML is interpreted as a unitary matrix by $[\![E]\!]$ in Figure 4.3b. We let h_M translate program M into Hermitian matrices by evaluating the expressions. Then $[\![E]\!]$ is the product of unitary matrices $e^{-i\tau h_M}$, each representing the solution to the Schrödinger equation under $H(t) = h_M$ for time duration τ. This is the solution to the Schrödinger equation governed by the piecewise-constant Hamiltonian programmed in E.

Implementation of HML

We implement HML in Python to ensure accessibility to physicists and other common users. For a quantum system, we store the sites in a list. A product Hamiltonian P is then stored as a list of site operators using the same order of the site list. We also employ a Python dictionary to store a time-independent Hamiltonian H where the key-value pairs are made of a product Hamiltonian P and its coefficient denoted by $H[P]$. Mathematically, $H[P] = \mathrm{Tr}\{H \cdot P\}$. We only store those P with non-zero $H[P]$ to compactly store Hamiltonians.

To deal with the algebraic operations of Hermitian matrices, we symbolically implement an algebraic group for site operators (the Pauli group), and then Hermitian expressions are evaluated accordingly. For example, $H_1 + H_2$ is effectively implemented by enumerating P appearing in the keys of H_1's and H_2's dictionary, and construct $(H_1 + H_2)[P] = H_1[P] + H_2[P]$. Another example is multiplication, where $H_1 \cdot H_2$ is implemented by enumerating P_j in H_j's dictionary keys. Since the site operators of different sites commute and those of the same sites are in a finite group, $P_1 \cdot P_2$ is a product Hamiltonian P with an additional scalar multiplier p (i.e., $(X_1X_2) \cdot (Y_1Y_2) = -1 \cdot Z_1Z_2$). We add $p \cdot H_1[P_1] \cdot H_2[P_2]$ to the coefficient $(H_1 \cdot H_2)[P]$. Then we represent the evolution E as a list of tuples (H, τ) encompassing Hermitian matrix H and the evolution time τ of an evolution segment.

Input Discretization Error In many-body physics systems, Hamiltonians are commonly continuous, taking form $H_{\text{tar}}(t) = \sum_{k=1}^{K} \alpha_k(t) H_k$. In HML, these Hamiltonians are discretized into a series of piecewise time-independent Hamiltonians in the input. Let the evolution duration be T and the discretization number be D. We discretize $H_{\text{tar}}(t)$ over time steps $\{t_d\}_{d=1}^{D}$ where $0 < t_1 < ... < t_D < T$ and use the left endpoint of each interval as its approximation. Formally, $H_{\text{tar}}(t)$ is approximated by

$$\tilde{H}(t) = \sum\nolimits_{k=1}^{K} \tilde{\alpha}_k(t) H_k, \qquad \tilde{\alpha}_k(t) = \sum\nolimits_{d=1}^{D} \alpha_k(t_d) \mathbb{1}_{[t_d, t_{d+1})}(t), \tag{4.3.1.1}$$

where $\mathbb{1}_{[a,b)}$ is the indicator function of set $[a, b)$. We assume $\|H_k\| = 1$ where $\|\cdot\|$ is the spectral norm of matrices, $\alpha_k(t)$ are piecewise M-Lipschitz functions, and $\{t_d\}_{d=1}^{D}$ include all partitioning points of the piecewise Lipschitz coefficients $\alpha_k(t)$. Then we can derive the error bound induced by discretization by the following lemma.

$$A \in \text{Site}, \quad v[q] \in \text{Var}^q \text{ for } q \in \{L, G\}, \quad r \in \mathbb{R}$$

$$\begin{aligned}
R \in \text{Operator} \quad &::= I \mid X \mid Y \mid Z \\
S^q \in \text{Para. Scalar}^q \quad &::= S_1^q + S_2^q \mid S_1^q \cdot S_2^q \mid S_1^q - S_2^q \mid S_1^q / S_2^q \\
&\quad \mid \exp(S^q) \mid \cos(S^q) \mid \sin(S^q) \mid r \mid v[q] \\
M^q \in \text{Para. Herm.}^q \quad &::= M_1^q + M_2^q \mid M_1^q \cdot M_2^q \mid S^q \cdot M^q \mid A.R \\
D \in \text{Device} \quad &::= M^G \mid M^L; D
\end{aligned}$$

(a) Syntax of AAIS specification language.

$$\begin{aligned}
h_{A.R} &= R_A, \\
h_{S^q \cdot M^q} &= \overline{\text{eval}}(S^q) \cdot h_{M^q}, \\
h_{M_1^q + M_2^q} &= h_{M_1^q} + h_{M_2^q}, \\
h_{M_1^q \cdot M_2^q} &= h_{M_1^q} \cdot h_{M_2^q}, \\
\{\!|M^G|\!\} &= h_{M^G}, \\
\{\!|M^L; D|\!\} &= h_{M^L}; \{\!|D|\!\}
\end{aligned}$$

(b) Semantics of AAIS programs.

Figure 4.4: Abstract syntax and denotational semantics of AAIS-SL. Here Site contains the sites of the device. Var^L and Var^G contain the local and global variables correspondingly. $\overline{\text{eval}}(S)$ evaluates S as a real function.

Lemma 4.3.1 ([14])**.** *The difference between the unitary $U(T)$ of evolution under $H_{\text{tar}}(t)$ for duration T and the unitary $\tilde{U}(T)$ of evolution under $\tilde{H}(t)$ is bounded by*

$$\left\| U(T) - \tilde{U}(T) \right\| \leq C_1 D^{-1} M K T^2. \tag{4.3.1.2}$$

Here $C_1 > 0$ is a constant, D is the discretization number, K is the number of terms in $H_{\text{tar}}(t)$, and L is the Lipschitz constant for $\alpha_k(t)$.

This lemma shows that when we increase the discretization number D, the evolution error in the approximation can be arbitrarily small, justifying the discretization. The proof is routine in quantum information and hence omitted.

4.3.2 AAIS Specification Language

To specify AAISs with programs, we propose and implement AAIS Specification Language (AAIS-SL) and present its abstract syntax and denotational semantics in Figure 4.4.

Abstract Syntax of AAIS-SL

To characterize the Hamiltonians of instructions, sites are declared with identifiers stored in a set Site, and site operators are defined as objects of sites by default.

Compared to HML, the major difference in the syntax is variables. Two types of variables whose identifiers are stored in Var^G and Var^L represent global variables and local variables, respectively. They are terms in parameterized scalars and consist of parameterized Hermitians. Then an AAIS for a device is effectively a collection of instruction Hamiltonians as parameterized Hermitian matrices, along with the system Hamiltonian.

Denotational Semantics of AAIS-SL

We interpret an AAIS D characterizing a device as a list of instructions along with the system Hamiltonian. Similar to the HML semantics, we employ a translation h for expressions S to obtain parameterized Hermitians. Function $\overline{\text{eval}}$ evaluates a parameterized scalar expression S as a real function taking a valuation of variables and outputting a real number. Hence h translates parameterized Hermitian expressions to Hamiltonians in the form of (4.2.0.1). Without ambiguity, we use $\{\!|\eta|\!\}(\vec{v})$ to represent the instruction Hamiltonian of η.

Implementation of AAIS-SL

We also provide a Python implementation of AAIS-SL. We store sites and Hermitian matrices similarly to the implementation of HML. The difference is that instead of storing real numbers as coefficients, we store Python functions taking global variable and local variable valuations as inputs. We build function algebraic operations (i.e., $(f_1 + f_2)(x) = f_1(x) + f_2(x)$ and $(f_1 \cdot f_2)(x) = f_1(x) \cdot f_2(x)$) to deal with expressions and establish the parameterized Hermitian matrix expressions. As described in Figure 4.4, an AAIS is effectively represented by a system Hamiltonian and a list of instruction Hamiltonians.

4.3.2.1 Examples of AAIS

Through AAISs, we provide a general framework to characterize the programmability of analog quantum simulators. Here we show how we design AAISs for QuEra, IonQ, and IBM devices. The design of AAIS abstraction pursues a balance between expressiveness and implementation hardness on real devices: to simulate more complicated quantum systems, more complicated instructions are needed, requiring more advanced technologies in their implementation.

Rydberg AAIS However, current QuEra devices do not support local laser addressing, meaning only a global laser interacts with every atom simultaneously. We use a variant for QuEra devices, called the global Rydberg AAIS, where there is only one instruction η with the instruction Hamiltonian

$$\{\!|\eta|\!\}(\Delta, \Omega, \phi) = -\Delta \sum\nolimits_{j=1}^{m} \hat{n}_j + \frac{\Omega}{2} \sum\nolimits_{j=1}^{m} (\cos(\phi) X_j - \sin(\phi) Y_j). \tag{4.3.2.1}$$

Heisenberg AAIS The IonQ and IBM devices, though using different platforms, share similar capabilities for constructing interactions. For both platforms, the Heisenberg AAIS is designed and implemented, which contains 1-site instructions $\eta_{j,P}$ and 2-site instructions $\eta_{j,k,PP}$ for $j, k \in \{1, ..., n\}$ and $P \in \{X, Y, Z\}$, where n is the number of sites. Each instruction possesses one local variable, and their instruction Hamiltonians are:

$$\{\!|\eta_{j,P}|\!\}(a) = a \cdot P_j, \qquad \{\!|\eta_{j,k,PP}|\!\}(a) = a \cdot P_j P_k. \tag{4.3.2.2}$$

Here, the 1-site instructions $\eta_{j,P}$ are defined for every site in the system, and the 2-site instructions $\eta_{j,k,PP}$ are only defined when $(j,k) \in E$ for an undirected connectivity graph E representing the connectivity of the detailed device. For ion trap devices, E is a complete graph with an edge between each site pair. Superconducting devices typically have limited connectivity, and we let E be the connectivity graph of the IBM devices.

The Heisenberg AAIS can simulate a family of Heisenberg models [51] covering the Ising models. A variant of the Heisenberg AAIS called the 2-Pauli AAIS extends the 2-site interactions to $P_j Q_k$ interactions for $P, Q \in \{X, Y, Z\}$, is capable of simulating more quantum systems, and is realizable on IonQ and IBM devices with specific connectivity.

IBM-Native AAIS Besides the Heisenberg AAIS, for the IBM devices, we can also model their native effects in an IBM-native AAIS. Its 2-site instructions are $\eta_{j,k,CR}$ for $(j,k) \in E$ where

$$\{|\eta_{j,k,CR}|\}(\Omega) = \omega_{ZX}\Omega Z_j X_k + \omega_{ZZ} Z_j Z_k + \omega_{IX}\Omega X_k + \omega_{ZI}\Omega^2 Z_j, \tag{4.3.2.3}$$

where $\omega_{ZX}, \omega_{ZZ}, \omega_{IX}$, and ω_{ZI} are device-dependent constants. Instruction $\eta_{j,k,CR}$ and $\eta_{l,k,CR}$ can be simultaneously executed on IBM devices because of platform features. However, since it contains multiple terms with limited freedom of control, only a few quantum systems can be directly simulated by the IBM-native AAIS. For the systems that can be simulated, a much shorter pulse duration can be produced. A more detailed analysis is in Section 4.5.2.

4.4 SimuQ compiler with intermediate representations

Compiling a target quantum system to an analog quantum simulator is computationally hard in most cases, especially when we aim at a general framework. In this section, we build the

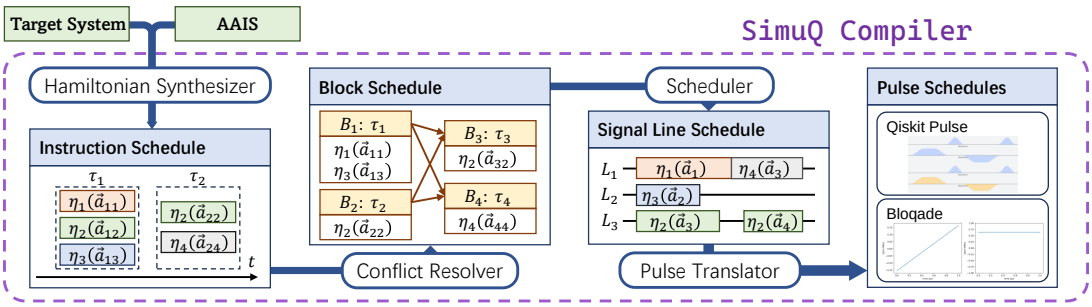


Figure 4.5: An illustration of the SimuQ compilation process.

first compiler for quantum simulation on general analog quantum simulators and several novel intermediate representations to conquer various challenges in the overall compilation.

The overall compilation workflow is presented in Figure 4.5. Since this is the first exploration of compilation to heterogeneous analog quantum simulators, our proposal intuitively decomposes the problem into several natural sub-problems that are rarely encountered in prior works and applies straightforward solutions to each step. Much space for optimizing our workflow within the scope of our approach is left for future work, which is left as future work.

4.4.1 Instruction schedules and Hamiltonian synthesizer

The first intermediate representation is instruction schedules that describe the execution of instructions on the device. We will also introduce a Hamiltonian synthesizer to create an instruction schedule that simulates a target quantum system.

4.4.1.1 Instruction Schedules

An instruction execution $(\eta, \vec{a}, \tau_s, \tau_e)$ specifies an instruction η, a valuation $\vec{v} \mapsto \vec{a}$ of η's local variables, and evolution starting time τ_s and ending time τ_e. It applies a Hamiltonian $H_\eta(\vec{a})$ to the device during $[\tau_s, \tau_e)$. In later cases when the absolute starting time and ending time are

unimportant, we also use duration $\tau_d = \tau_e - \tau_s$ in instruction executions.

An instruction schedule includes a valuation $\vec{g}$ to the global variables and a set of instruction executions $\{(\eta_j, \vec{a}_j, \tau_{s,j}, \tau_{e,j})\}$. At the time t, the instruction executions satisfying $\tau_{s,j} \leq t < \tau_{e,j}$ generate effects on the device. Executing the instruction schedule evolves the device, governed by:

$$H(t) = H_{\text{sys}}(\vec{g}) + \sum\nolimits_{j:\tau_{s,j} \leq t < \tau_{e,j}} H_{\eta_j}(\vec{a}_j). \tag{4.4.1.1}$$

We use a more succinct representation of the instruction schedules generated by our Hamiltonian synthesizer. We characterize the set of instruction executions as a list $\mathcal{S} = [(C_j, \tau_j)]_{j=1}^m$ where $C_j = \{(\eta_{jk}, \vec{a}_{jk})\}_k$. $\mathcal{S}$ denotes a sequential evolution of simultaneous instruction executions in C_j for time duration τ_j. Let $T_j = \sum_{k \leq j} \tau_j$ and assume $T_0 = 0$. The absolute starting and ending time of instruction execution $(\eta_{jk}, \vec{a}_{jk}) \in C_j$ are then T_{j-1} and T_j. Mathematically, the Hamiltonian $H(t)$ governing the evolution of the device at time $t \in [T_{j-1}, T_j)$ is $H(t) = H_{\text{sys}}(\vec{g}) + \sum_k \{|\eta_{jk}|\}(\vec{a}_{jk})$. As a solution to the Schrödinger equation, the execution of instruction schedule $(\mathcal{S}, \vec{g})$ results in an evolution of the device described by a unitary matrix

$$U(T_m) = \prod\nolimits_{j=m}^{1} e^{-i\tau_j \left(H_{\text{sys}}(\vec{g}) + \sum_k \{|\eta_{jk}|\}(\vec{a}_{jk})\right)}. \tag{4.4.1.2}$$

4.4.1.2 Quantum Simulation by Executing Instruction Schedules

We formally define the task of compiling quantum simulations to a quantum device described by an AAIS. Consider a target quantum system described by a Hamiltonian $H_{\text{tar}}(t)$ and evolution time interval $[0, T)$. Compilation of a quantum simulation asks for a site layout L and

an instruction schedule $(\mathcal{S}, \vec{g})$. A site layout L is an injective mapping from each site in the target system to a site in the device system. We call the Hilbert space of the sites mapped to by L the layout subspace of the device Hilbert space. A layout L induces a mapping $\mathcal{L}$ from the target system's Hilbert space to the layout subspace. When limiting $\mathcal{L}(H)$ in the layout subspace where H is a Hermitian matrix in the target Hilbert space, one can relabel the sites of $\mathcal{L}(H)$ according to L^{-1} and recover H. When $H(t)$ is a time-dependent Hamiltonian of the target Hilbert space, we write $\mathcal{L}(H)$ as a Hamiltonian of the device Hilbert space satisfying $\mathcal{L}(H)(t) = \mathcal{L}(H(t))$. Let the execution of the instruction schedule $(\mathcal{S}, \vec{g})$ produce a unitary matrix U and let the evolution under $\mathcal{L}(H_{\text{tar}})$ for time interval $[0, T)$ be $\mathcal{L}(U_{\text{tar}})$. We say that a site layout L and the instruction schedule $(\mathcal{S}, \vec{g})$ simulate $H_{\text{tar}(t)}$ if U approximates $\mathcal{L}(U_{\text{tar}})$.

4.4.1.3 Hamiltonian Synthesizer

Since HML discretizes continuous Hamiltonians with small errors, in this step, we consider a target quantum system described by a sequence of evolution under $H_{\text{tar},j}$ for time duration τ_j indexed by $j \in \{1, ..., N\}$. We want to synthesize an instruction schedule simulating the target quantum system on a device described by an AAIS $D = [\eta_1; ...; \eta_M; H_{\text{sys}}]$. Our Hamiltonian synthesizer follows a three-step loop: (1) propose a site layout L; (2) build a coefficient equation system; (3) solve the mixed-binary equation system. If the solver does not find an approximate solution, we repeat this process until a timeout condition is met.

Site layout proposer The first step of the synthesizer loop proposes a site layout L and later steps check its feasibility. To the best of our knowledge, although layout synthesis for quantum circuits is thoroughly studied [52], there is no prior work on the layout synthesis for Hamiltonian-oriented

quantum computing. The main difference between them is the unavailability of swap gates for many analog quantum devices, i.e., QuEra's Rydberg atom arrays.

We employ a search with pruning as a general solution to a layout proposer. The pruning strategy is to abort the search when there exists a product Hamiltonian P and j where $H_{\text{tar},j}[P] \neq 0$ and $\mathcal{L}(P)$ does not have a non-zero coefficient expression in any η_k and H_{sys}. This abort condition can be met halfway through the search. For a partial layout L (where several sites are not assigned in L yet) and a product Hamiltonian P, we can map it to a product Hamiltonian $\mathcal{L}(P)$ of the device with holes on several sites. When searching for $\mathcal{L}(P)$ in an AAIS, holes can match any site operator. If none is found, the current search branch is aborted.

After proposing a layout, we proceed to steps (2) and (3) to check its feasibility. If rejected, the above search process returns and proceeds to other search branches to propose another layout. If all possibilities are not feasible, the compiler will report no solution and fail the process.

Coefficient equation builder We synthesize instruction executions by a system of mixed-binary non-linear equations to match coefficients of product Hamiltonian in the target quantum system.

Given a site layout L, a set of equations is constructed to match the coefficients in $H_{\text{tar},j}$ for every $1 \leq j \leq N$. We create time variables t_j to represent the evolution time for instruction executions synthesizing evolution of $H_{\text{tar},j}$ for time τ_j, with constraints $t_j > 0$. For instruction η_k in the AAIS, we create an indicator variable $s_{k,j} \in \{0, 1\}$ to indicate whether η_k is selected to be executed in the synthesis of $H_{\text{tar},j}$. Assuming that η_k has local variables $\vec{v}_k$ of dimension $|\vec{v}_k|$, we create $|\vec{v}_k|$ new equation system variables stored in a vector $\vec{a}_{k,j}$. For global variables, we create a vector $\vec{g}$ of dimension $|\vec{v}_{\text{glob}}|$ of the AAIS, which is independent of j. In total, we have created $|\vec{v}_{\text{glob}}| + N \sum_k |\vec{v}_k| + N$ real variables for global variables, local variables, and time variables

Algorithm 1 Equation builder for Hamiltonian synthesis.

Inputs: site layout mapping $\mathcal{L}$, target quantum system $(H_{\text{tar},j}, \tau_j)$ for $1 \leq j \leq N$, AAIS $D = [\eta_1, ..., \eta_M, H_{\text{sys}}]$, equation system variables $\{\vec{a}_{k,j}\}, \{s_{k,j}\}, \vec{g}, \{t_j\}$

Output: a system of equations Υ

```
Υ ← {}
for j ∈ {1, ..., N} do
    G ← {L(H_tar,j)}_{j=1}^N ∪ {H_sys}
    Q ← [P | ∃H ∈ G, H[P] ≠ 0]
    i ← 0
    while i < |Q| do
        P ← Q.getitem(i)
        i ← i + 1
        if P = I then
            continue
        e ← H_tar,j[P](g⃗)
        for k ∈ {1, ...M} do
            if {|η_k|}[P] ≢ 0 then
                e ← e + {|η_k|}[P](a⃗_k,j) · s_k,j
                for P′ ∉ Q : {|η_k|}[P′] ≢ 0 do
                    Q.append(P′)
        Υ.add(t_j · e = τ_j · L(H_tar,j)[P])
```

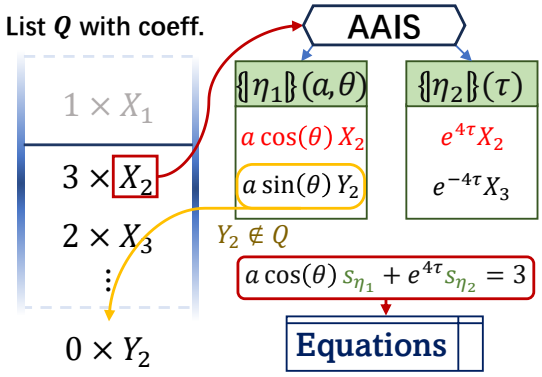


Figure 4.6: An example illustrating the equation builder. X_2 is searched for in AAIS, where η_1 and η_2 are found and used in equation for X_2.

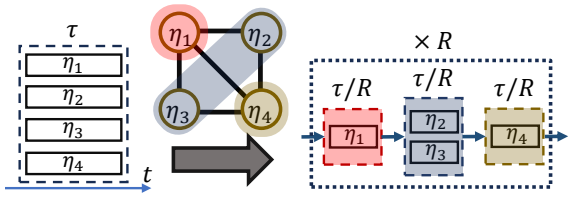


Figure 4.7: An example of Trotterization from an instruction schedule to a block schedule, with a conflict graph and its grouping.

respectively, and NM indicator variables.

Then we establish a coefficient equation for each product Hamiltonian P to match $H_{\text{tar},j}$:

$$(\forall j), (\forall P \neq I): \quad t_j \cdot H_{\text{sys}}(\vec{g})[P] + \sum_{k=1}^{M} t_j \cdot \{|\eta_k|\}[P](\vec{v}_{k,j}) \cdot s_{k,j} = \tau_j \cdot \mathcal{L}(H_{\text{tar},j})[P]. \tag{4.4.1.3}$$

Here the left-hand-side calculates the summed effects of P from each instruction and the system Hamiltonian and the right-hand-side calculates the effect of P in the target quantum system.

There are typically many trivial equations in this system having 0 on both sides since only a few P appear in either $H_{\text{tar},j}$ or $\{|\eta_k|\}$ with respect to the exponentially many possible combinations of site operators. We propose Algorithm 1 to find all non-trivial equations. This algorithm

starts with a list Q containing all the product Hamiltonians with non-zero coefficients in H_{sys} and $\mathcal{L}(H_{\text{tar},j})$. It then enumerates the list Q and establishes coefficient equations for each P by enumerating instructions η_k in AAIS. During this process, it may encounter instruction Hamiltonians $\{\!|\eta_k|\!\}$ who contain product Hamiltonians P' that never appears in Q. These product Hamiltonians may lead to non-trivial equations, so we add them to Q. An example of this procedure is illustrated in Figure 4.6.

Mixed equation solver The established coefficient equation system is mixed-binary and non-linear. A solver is applied to obtain approximate solutions which correspond to instruction schedules.

We provide several options for the solver. The first is dReal [53] based on δ-complete decision procedures, which supports real variables, binary variables, and algebraic functions in HML and AAIS-SL. It performs well when the coefficient expressions are close to linear (the Heisenberg AAIS), while poorly when highly non-linear (the Rydberg AAIS).

As another option, we construct a least-squares-based solver. This solver uses a relaxation-rounding scheme. We apply a continuous relaxation to loosen the value range of indicator variables from $s_{k,j} \in \{0, 1\}$ to $\hat{s}_{k,j} \in [0, 1]$, substitute $\hat{s}_{k,j}$ for $s_{k,j}$ in the equation system, and solve the equation system by least-squares methods via an implementation in SciPy [54]. We then round the indicator variables $s_{k,j}$ according to the solution. The criterion sets $s_{k,j}$ to 1 if there is $\sum_P t_j \{\!|\eta_{k,j}|\!\}[P](\vec{v}_{k,j})\hat{s}_{k,j} > \delta$ for a pre-defined tolerance parameter δ, and sets to 0 otherwise. This criterion evaluates how much error the solution will induce if we set $s_{k,j}$ to 0. We then solve the equation system again to obtain a more precise solution.

The solver generates an approximate solution with error e, defined by

$$e = \sum\nolimits_{k,j,P} |\tau_j \{|\eta_k|\}[P](\vec{v}_{k,j}) - t_j \mathcal{L}(H_{\mathrm{tar},j})[P]|. \tag{4.4.1.4}$$

If $e < \epsilon$ where ϵ is a pre-defined tolerance, the solution is accepted. Otherwise, we return to step (1) to generate another layout and check feasibility. An accepted solution induces an instruction schedule $(\mathcal{S} = \{(C_j, t_j)\}, \vec{g})$ where $C_j = \{(\eta_k, \vec{a}_{k,j}) : s_{k,j} = 1\}$.

4.4.1.4 Error Induced by Hamiltonian Synthesizer

Now we bound the error in the evolution induced by the approximation in the equation solving of the Hamiltonian synthesizer since our solver generates approximate numerical solutions. Let $\hat{U}(T)$ be the unitary of executing generated instruction schedule $(\mathcal{S}, \vec{g})$, and $\tilde{U}'(T) = \mathcal{L}(\tilde{U}(T))$ be the unitary of the evolution of the discretized target system after site layout mapping $\mathcal{L}$. We can conclude the error induced by the Hamiltonian synthesizer is bounded by tolerance ϵ in the equation solving and the proof is routine and omitted.

Lemma 4.4.1 ([14])**.** *The error of evolution induced by equation solving is bounded by a constant $C_2 > 0$ and error bound ϵ with the following inequality:*

$$\left\| \tilde{U}'(T) - \hat{U}(T) \right\| \leq C_2 \epsilon. \tag{4.4.1.5}$$

Remark 4.4.1. *In general, compiling a target system is computationally hard. Finding a site layout for machines with specific topology can be as hard as the sub-graph isomorphism problem, an NP-complete problem. Besides, since the design of AAIS does not pose strict restrictions on*

the expressions, pathological functions may emerge in the coefficients, which complicates the equation-solving process. Our solutions to these problems may not be optimal but are intuitive, feasible, and efficient enough for most cases (also refer to Section 4.5 for detailed case studies).

4.4.2 Block schedules and conflict resolver

Instruction schedules are oversimplified descriptions of what can be executed on the devices. Mainly, there are two realistic restrictions not captured by instruction schedules. First, some instructions on real devices can not be executed simultaneously. For example, on an IonQ device, $\eta_{1,2,XX}$ cannot be simultaneously executed with $\eta_{1,2,ZZ}$ since they use the same interaction process with different bases. Second, instruction execution implementations may take longer than the scheduled execution time. We propose a flexible generalization to the instruction schedules called block schedules and implement a conflict resolver to compile generated instruction schedules to block schedules.

4.4.2.1 Block Schedules

A block schedule is a temporal graph whose vertices are blocks of instruction executions, together with the valuation of the global variables. An instruction block B contains a collection of instruction executions whose evolution duration is τ. The block schedule is then a directed acyclic graph where an edge $(B_j \to B_k)$ is a restriction: instructions in B_k should start simultaneously after instruction executions in B_j end. Instruction schedules generated by our Hamiltonian synthesizer are special cases of block schedules where the temporal graph forms a chain and blocks are the collections of instruction executions.

When executing a block schedule, we first decide the execution order $\gamma : (B_1, ..., B_r)$ of the blocks and then evolve the system by γ sequentially. Let the B_j contain $\{(\eta_{j,k}, \vec{a}_{j,k})\}_k$ with evolution time τ_j. The evolution will generate a unitary transformation

$$U_\gamma = \prod_{j=r}^{1} e^{-i\tau_j(H_{\text{sys}}(\vec{g})+\sum_k \{|\eta_{j,k}|\}(\vec{a}_{j,k}))}. \tag{4.4.2.1}$$

Our next step is to generate a block schedule where instructions in each block are simultaneously executable and approximate the execution of the instruction schedule.

4.4.2.2 Instruction Decorations

In general, the conflict relation of instructions forms a graph F: $(\eta_j, \eta_k) \in F$ means that η_j and η_k cannot be executed simultaneously. To ease the description of F, we introduce decorations to instructions to specify properties like categories of instructions. More decorations can be added based on the detailed hardware restrictions accordingly.

Signal Lines Physical pulses are sent to devices through signal carriers like electronic wires or arbitrary waveform generators (AWG). A natural conflict is that if two instructions require the same signal carrier, they cannot be executed simultaneously. We abstract the concept of signal carriers as *signal lines* and assign each instruction η to a signal line denoted by $SL(\eta)$. If $SL(\eta_j) = SL(\eta_k)$, instructions η_j, η_k conflict with each other.

Nativeness Another aspect leading to conflicts is whether instruction implementations employ compound pulses to approximate an effective Hamiltonian. For example, IBM devices generate $\{|\eta_{j,k,CR}|\}$ by direct microwave controls of a cross-resonance pulse [55]. Hence the IBM-native AAIS for IBM devices has $\eta_{j,k,CR}$ as *native* instructions: they can be simultaneously executed

with other native instructions. To effectively realize $\{\!|\eta_{j,k,ZZ}|\!\}$, a compound sequence of microwaves including two cross-resonance pulses is applied to approximate a $Z_j Z_k$ interaction [56]. Simultaneously applying other pulses on site j or k will break the approximation. Hence $\eta_{j,k,ZZ}$ are *derived* instructions in the IBM-native AAIS.

Let $\mathsf{inf}(H)$ be the sites on which Hamiltonian H acts non-trivially (when limited on these sites, H is not identity). We assume that implementing a derived instruction η only affects $\mathsf{inf}(\{\!|\eta|\!\})$. Then a derived instruction η_1 conflicts with η_2 if $\mathsf{inf}(\{\!|\eta_1|\!\}) \cap \mathsf{inf}(\{\!|\eta_2|\!\})$ is not empty.

4.4.2.3 Conflict Resolver via Trotterization

Given a conflict graph and an instruction schedule $(S, \vec{g})$, we implement a conflict resolver to generate a block schedule without conflicts in each block.

A well-studied technique in quantum information to simulate summed Hamiltonians in quantum simulation is Trotterization. Let Hamiltonian $H = \sum_{j=1}^{L} H_j$ where L Hamiltonians evolve the system simultaneously. We assume we have a device supporting evolving single H_j for any duration t, realizing unitary matrix e^{-itH_j}, while there is no evolution under $\sum_{j=1}^{L} H_j$. Trotterization (also known as the product formula algorithm) [57] makes use of the Lie-Trotter formula

$$e^{-it\sum_j H_j} = \lim_{n\to\infty} \left(\prod_j e^{-i\frac{t}{n}H_j}\right)^n \approx \left(\prod_j e^{-i\frac{t}{N}H_j}\right)^N. \tag{4.4.2.2}$$

By choosing a large N, the above formula shows that we can approximate the evolution under H for time T by repeating for N times a sequential evolution for $j \in \{1, ..., L\}$ under H_j for time t/N. Each segment of evolution realizes a unitary transformation $e^{-i(t/N)H_j}$ as in the formula.

First, we consider the case where $H_{\text{sys}} = 0$. Each (C_d, τ_d) in $\mathcal{S}$ is considered independently. Let $C_d = \{(\eta_j, \vec{a}_j)\}_j$ and the conflict graph of these instructions be F. To accommodate Trotterization in a conflict resolver, we first categorize the instructions into groups without conflict. The grouping is effectively a coloring of vertices in F where no edge connects monochromatic vertices. We employ a greedy graph coloring algorithm from NetworkX [58] to find a feasible grouping $\{G_j\}_{j=1}^{L}$ with L colors where G_j contains instruction executions in the j-th group.

A temporal graph in a block schedule can depict the process in (4.4.2.2). Let H_j be the Hamiltonian of simultaneous instruction executions in G_j, $H_j = \sum_{(\{\eta_k\},\vec{a}_k)\in G_j} \{\eta_k\}(\vec{a}_k)$, and R be the Trotterization number specified by users. An evolution of H_j for time τ_d/R corresponds to a block $B_j = (G_j, \tau_d/R)$. Then a sequential evolution of $\{H_j\}_{j=1}^{L}$ forms a chain $B_1 \to \cdots \to B_L$. We create R copies of this chain and connect them sequentially to represent the Trotterization process.

Additionally, we deal with the cases where the system Hamiltonian H_{sys} is non-zero. Let $\tilde{L}$ be the maximal coloring number L in the above process. We assume that there exists $\vec{g}_{\tilde{L}}$ such that $H_{\text{sys}}(\vec{g}_{\tilde{L}}) = H_{\text{sys}}(\vec{g})/\tilde{L}$. Some devices may not support this assumption, but it is rarely used since only a few devices with non-zero system Hamiltonian have conflicting instructions. We then augment the number of groups to $\tilde{L}$ for each $(C_j, \tau_j) \in \mathcal{S}$ by adding empty sets in groupings. Now we create a block schedule with $\vec{g}_{\tilde{L}}$ and a temporal graph constructed on the augmented groupings. Executing this block schedule approximates the execution of the given instruction schedule.

4.4.2.4 Error Induced by Conflict Resolver

The Trotterization resolves conflicts while also introducing errors. We denote the instruction schedule where $S = \{(\{(\eta_{d,j}, \vec{a}_{d,j})\}_j, \tau_d)\}_{d=1}^{D}$ and its evolution as $\hat{U}(T)$. For segment d of evolution in S, we assume the grouping is $\{G_j^d\}_{j=1}^{L_d}$ and the evolution by executing the block schedule as $\bar{U}(T)$.

Lemma 4.4.2 ([59])**.** *The difference between $\hat{U}(T)$ and $\bar{U}(T)$ of evolution after resolving conflicts by Trotterization is bounded by*

$$\left\|\hat{U}(T) - \bar{U}(T)\right\| \leq \frac{(\Lambda T)^2}{DR} e^{\frac{\Lambda T}{DR}}. \tag{4.4.2.3}$$

Here $\Lambda = \max_{d,j} L_d \left\|\sum_{(\eta,\vec{a})\in G_j^d} \{|\eta|\}(\vec{a})\right\|$, D and R are the discretization and Trotterization numbers.

As implied by this lemma, in ideal cases, increasing the Trotterization number reduces the induced error to arbitrarily small. However, it also increases the total number of instruction executions. Due to the non-negligible error accumulations in each instruction execution on devices, there is a trade-off over the Trotterization number R depending on the real-time parameters of the device, where we leave the freedom to user specification.

Although Trotterization provides a theoretical guarantee of dealing with conflicts [60] with bounded approximation errors, in practice, a large Trotterization number can also amplify device noises. Many optimization techniques may be used in Trotterization, and we implement the following ones as a demonstration. We leave further optimizations as future directions.

- Order inside a Trotterization step. Notice that the error bound in [59] does not require a

fixed simulation order of H_j. We realize the freedom here by setting the blocks in one Trotterization step to be parallel in the temporal graph and connecting the blocks in the next step after the blocks in this step. This freedom may be exploited in the following compilation passes.

- Blocks commuting with others. If the Hamiltonian of a block commutes with other blocks, we separate it from others, evolve it first, and resolve the conflicts of others. This procedure does not introduce errors.

- High-order Trotterization. We also integrated the second-order Trotterization method in our compiler, whose practical performance archives a balance between approximation errors and machine errors.

4.4.3 Signal line schedules and scheduler

The eventual output of SimuQ contains the pulses sent through signal carriers for devices to execute. We propose another intermediate representation, called a *signal line schedule*, to depict the concrete instruction executions sent through each signal line abstracted in Section 4.4.2.2 before generating platform-dependent executable pulses. For signal line l, it contains a list of instruction executions $(\eta, \vec{a}, \tau_s, \tau_e)$ with absolute starting and ending times and satisfying $SL(\eta) = l$.

To obtain a signal line schedule, we build a scheduler to traverse the temporal graph of the block schedule via a topological sort and generate a valid execution order of block schedules. It employs a first-arrive-first-serve principle for each signal line. The scheduler first extracts information about how long implementing each instruction execution takes from real devices.

Next, it arranges instruction executions on the signal lines at the earliest possible starting time obeying the order.

Remark 4.4.2. *The scheduling process may be independently configured and optimized, and the scheduler may use other criteria to determine the traversal order of instruction blocks or the alignment of blocks within the scheduled order as long as the hardware permits. This freedom in the scheduling process may be leveraged to reduce cross-talk [61] between the blocks or save small implementation overheads. We illustrate only a basic strategy and leave the exploitation for the future.*

4.4.4 Pulse schedules and pulse translator

In its final stage, the SimuQ compiler translates a signal line schedule into a pulse schedule using hardware providers' domain languages and APIs.

We extract the pulse shapes from the devices for each platform to implement instruction executions. We substitute the instruction execution on each signal line for pulse shapes configured by the valuations of local variables via the format specified by a pulse-enabled quantum device provider.

4.4.4.1 Translation to Hardware APIs

There are few pulse-enabled quantum device providers, and programming pulses is a challenging endeavor that requires extensive platform knowledge of various hardware and software engineering considerations. We demonstrate the effectiveness of SimuQ using QuEra, IBM, and IonQ devices.

QuEra's Rydberg atom devices Two APIs to QuEra devices are supported by SimuQ for the global Rydberg AAIS: Bloqade [41] programs and Amazon Braket programs. We set the atom positions according to the valuation of global variables and laser configurations as piecewise constant functions according to the valuations of local variables. Since the detuning Δ and amplitude Ω generate linear effects, piecewise linear laser configurations are also supported as an option. For Amazon Braket programs, $\Omega(t)$ should start and end at amplitude 0, so we add short (0.1ms) time intervals to the pulses' beginning and end with linear ramping. We also scale the pulse schedules to a total length of around 3.5ms to fit in the 4ms duration limit of the device.

IBM's superconducting devices For IBM devices, SimuQ can generate Qiskit Pulse programs for the Heisenberg AAIS and the IBM-native AAIS. For single-site instructions, the IBM device supports implementations of native X and Y instructions and derived Z instructions. We build up DRAG pulses [62] to realize X and Y instructions and free Z rotations [63] to realize Z instructions, which are standard superconducting device techniques. Two-qubit instructions in the Heisenberg AAIS are realized through the $Z_j X_k$ interactions created by echoed cross resonance pulses [55] together with single-qubit evolution to change bases. We follow Earnest et al. [64] and realize interaction-based gate implementations, whose benefits are further explained in Section 4.5.3. Additionally, we extract cross-resonance pulses from Qiskit and compose pulses to realize native $\eta_{j,k,CR}$ in the IBM-native AAIS.

IonQ's trapped-ion devices SimuQ supports both IonQ cloud and Qiskit circuit programs for IonQ devices with the Heisenberg AAIS. Unlike QuEra and IBM devices, IonQ does not provide pulse-level programmability for their ion trap devices. However, we can still exploit their native gate set to generate a quantum circuit with precise control of the execution on their devices. With

the support of partially entangling Mølmer-Sørenson gate [65], we can implement instructions of the Heisenberg AAIS with higher fidelity. More details are explained in our case studies in Section 4.5.3.

4.4.4.2 Semantics of Pulse Schedules and Errors in Instruction Implementation

Abstractly, a pulse schedule includes a time-dependent function $\vec{f_l}(t)$ (pulses) for signal line l, generating the effective Hamiltonian $H_l(t)$ physically. For example, instruction execution $(\eta, \vec{a}, \tau_s, \tau_e)$ for signal line l in the signal line schedule should be translated into pulses $\vec{f_l}(t)$ that effectively generate $H_l(t) = \{\!|\eta|\!\}(\vec{a})$ for $\tau_s \leq t < \tau_e$. Collectively, the Hamiltonian on the device is $H_{\text{dev}}(t) = H_{\text{sys}} + \sum_l H_l(t)$, and the semantics of executing a pulse schedule is the unitary evolution under $H_{\text{dev}}(t)$. Yet, the implementation of instructions on real devices may be imperfect. We assume that there is a implementation error threshold Δ such that the on-device $\check{H}_l(t)$ and $\check{H}_{\text{sys}}$ satisfies $\max_{t,l} \left\|\check{H}_l(t) - H_l(t)\right\| \leq \Delta$ and $\left\|\check{H}_{\text{sys}} - H_{\text{sys}}\right\| \leq \Delta$, forming on-device evolution under $\check{H}_{\text{dev}}(t) = \check{H}_{\text{sys}} + \sum_l \check{H}_l(t)$. Since the signal line scheduler does not alter the semantics of block schedules, we bound the implementation error on the device.

Lemma 4.4.3 ([14])**.** *The difference between the unitary $\check{U}(T)$ on the device and the unitary $\bar{U}(T)$ of executing the block schedule generated by the conflict resolver is bounded by*

$$\left\|\bar{U}(T) - \check{U}(T)\right\| \leq C_3 S \Delta \Gamma T, \tag{4.4.4.1}$$

where $C_3 > 0$ is a constant, S is the number of signal lines and system Hamiltonians, and $\Gamma = \max_d L_d$ is the maximal number of groups in the conflict resolver.

With a faithful implementation of instructions on real devices, the pulse translator pro-

duces negligible errors. The proof is routine in quantum information and is therefore omitted. We remark that other forms of device errors (e.g., high-energy space leakage) can be analyzed similarly.

4.4.5 Semantics preservation of SimuQ compiler

If compilation succeeds, the SimuQ compiler generates executable pulse schedules from programmed quantum systems with bounded errors. We conclude the approximate semantics preservation theorem of the SimuQ compilation process using Lemma 4.3.1, Lemma 4.4.1, Lemma 4.4.2, and Lemma 4.4.3.

Theorem 4.4.4 (Semantics Preservation)**.** *Given a Hamiltonian $H_{\text{tar}}(t) = \sum_{k=1}^{K} \alpha_k(t) H_k$ where α_k is piecewise M-Lipschitz and $\|H_k\| = 1$, if the compilation succeeds, SimuQ generates a site layout L and an executable pulse schedule. Let the unitary $U(T)$ represent the evolution under $H_{\text{tar}}(t)$ for duration $[0, T]$ and $\check{U}(T)$ for the evolution executing the pulse schedule on the device. We have*

$$\left\|\mathcal{L}(U(T)) - \check{U}(T)\right\| \leq C_1 D^{-1} M K T^2 + C_2 \epsilon + (\Lambda T)^2 D^{-1} R^{-1} e^{\frac{\Lambda T}{DR}} + C_3 S \Delta \Gamma T. \quad (4.4.5.1)$$

Here, T is the evolution time, $\mathcal{L}$ is the site layout mapping of layout L, C_1, C_2, C_3 are constants, D is the discretization number, ϵ is the error threshold in Hamiltonian synthesizer, R is the Trotterization number, Δ is the instruction implementation error threshold, S is the number of signal lines and system Hamiltonians on the device, and Λ and Γ depend on the Trotterization strategy in the compilation.

Tuning D, ϵ, R and improving the implementation to decrease δ can reduce errors induced

by the SimuQ compiler to arbitrarily small. The evolution under H_{tar} is hence simulated on the device.

4.5 Case studies

We conduct case studies highlighting SimuQ's portability. We also establish a small benchmark of quantum simulation to evaluate the SimuQ compiler performance.

4.5.1 Multiple-platform compatability

We compile and execute the Ising model on multiple supported devices of SimuQ. The following experiments show the portability of SimuQ on heterogeneous analog quantum simulators. We only need to program the target quantum systems once and apply the SimuQ compiler to generate code for different platforms and deploy and execute them on multiple real devices.

We focus on the simulation of the Ising model. We demonstrate two instances: a 6-site cycle and a 6-site chain, mathematically depicted by

$$H_{\text{chain}} = \sum\nolimits_{j=1}^{5} Z_j Z_{j+1} + \sum\nolimits_{j=1}^{6} X_j, \quad H_{\text{cycle}} = H_{\text{chain}} + Z_1 Z_6. \tag{4.5.1.1}$$

The target quantum system is to simulate H_{cycle} and H_{chain} for $T = 1$. When Trotterization is utilized, we set the Trotterization number to be 4, which is empirically selected based on experiment results.

SimuQ successfully compiles H_{cycle} on QuEra devices using the global Rydberg AAIS, both H_{cycle} and H_{chain} on IonQ devices using the Heisenberg AAIS, and H_{chain} on IBM devices using the Heisenberg AAIS. We send the generated code to execute on corresponding devices.

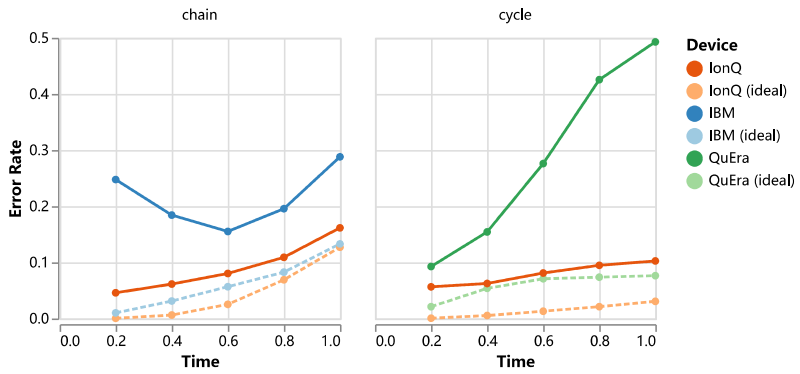


Figure 4.8: The simulation errors of the 6-site Ising models on multiple platforms. Ideal results are obtained by compiling with SimuQ and executing on noiseless simulators.

Since QuEra devices do not support state tomography, we evaluate the results on these platforms by a metric based on measurements supported by all devices in our experiments. We obtain the frequency of reading a bit-string s in a measurement instantly after the simulation finishes as a distribution $\mathbb{P}_{\text{exp}}[s]$ and numerically calculate the ground truth distribution $\mathbb{P}_{\text{GT}}[s]$ of obtaining s. We utilize the total variation distance $TV(\mathbb{P}_{\text{exp}}, \mathbb{P}_{\text{GT}}) = \frac{1}{2}\sum_{s\in\{0,1\}^6}|\mathbb{P}_{\text{exp}}[s] - \mathbb{P}_{\text{GT}}[s]|$ to evaluates the errors. We present the classical simulation of devices and real device execution results in Figure 4.8.

The minor errors in the classical simulations indicate the correctness of our framework. In ideal cases, the errors are induced by uncancellable non-neighboring Z_jZ_k interactions and short ramping times for the global Rydberg AAIS and by Trotterization errors for the Heisenberg AAIS. The real device execution results show higher errors than classical simulation because of device noises, while they are valid quantum simulation results. The errors on the IBM device are non-monotone, likely because the large state preparation and measurement errors affect more heavily the cases where the states deviate only a little from the initial state.

SimuQ fails to compile H_{chain} on QuEra devices since it requires different local detuning parameters for different sites, which current QuEra devices and the global Rydberg AAIS do not

support. It also fails to compile H_{cycle} on IBM devices since there is no 6-vertex cycle in IBM devices.

4.5.2 Hamiltonian-oriented compilation with native instructions

The most significant benefit of enabling Hamiltonian-level programming is to gain fine-grained and multi-site control via native operations. Near-term quantum devices have short coherence times: quantum states will deteriorate and lose their quantumness quickly. Generating shorter pulses to achieve the same effects is one of the crucial tasks for compilers of modern quantum devices. In this case study, we showcase the advantage in the lengths of pulse schedules enabled by Hamiltonian-oriented compilation using the IBM-native AAIS.

Our target quantum system evolves under $H_{2ZX} = Z_1X_2 + X_2Z_3$ for time $T = 1$, a small 3-site system. The IBM-native AAIS contains two native instructions $\eta_{1,2,CR}$ and $\eta_{3,2,CR}$ with Z_1X_2 and X_2Z_3 interactions respectively. Following Greenaway et al. [66], the simultaneous execution of them can be realized by simultaneously applying two cross-resonance pulses on IBM devices. By automatically compensating the other terms in SimuQ with native instructions $\eta_{2,X}$ and derived instructions $\eta_{1,Z}$ and $\eta_{2,Z}$ (their effects commute with $\{|\eta_{1,2,CR}|\}$ and $\eta_{3,2,CR}$ so no Trotterization is needed), the uncancellable remains are Z_1Z_2 interactions and Z_2Z_3 interactions. Fortunately, they can be reduced to one magnitude smaller than Z_1X_2 and X_2Z_3 interactions when selecting a relatively large Ω, and are considered small errors in the compilation. The pulse schedule to realize H_{2ZX}, displayed in Figure 4.9, is around 280ns long.

H_{2ZX} can also be compiled on IBM devices by a circuit-based compilation with the help of Qiskit. It first decomposes the simulation into a circuit with two gates $R_{Z_1X_2}(2)R_{X_2Z_3}(2)$ where

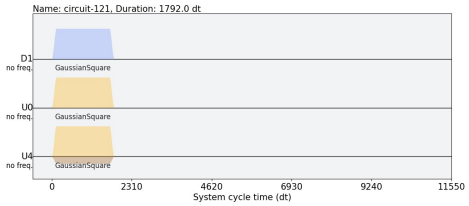


Figure 4.9: Pulses generated by SimuQ for evolution under H_{2ZX} for duration 1.

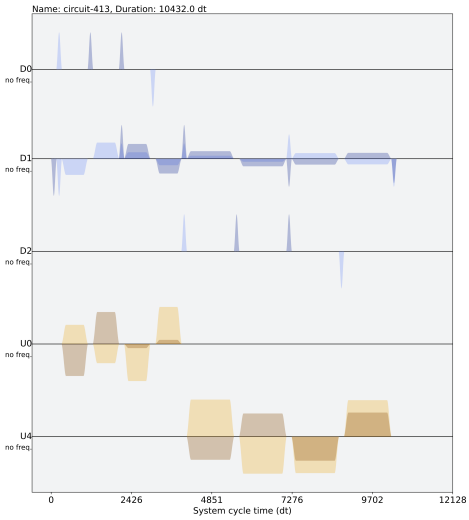


Figure 4.10: Pulses generated by Qiskit compiler for evolution under H_{2ZX} for duration 1.

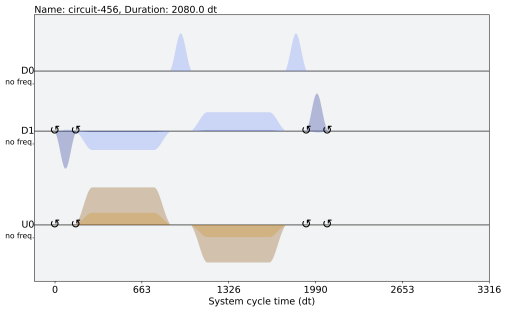


Figure 4.11: Pulses generated by SimuQ for evolving Z_0Z_1 for $T = 1$.

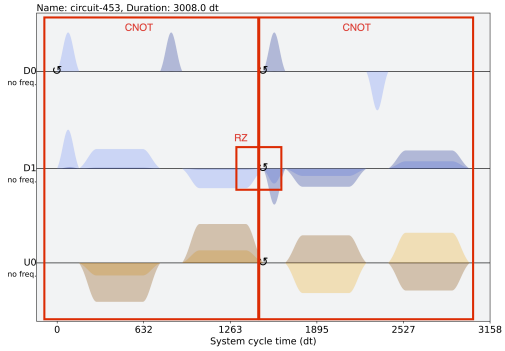


Figure 4.12: Pulses generated by Qiskit for evolving Z_0Z_1 for $T = 1$.

Table 4.2: The differences between the ideal $C(s)$ and the measured $C(s)$ on devices are displayed for different layers p with SimuQ and Qiskit CNOT-based compiler and are better when lower.

p	IBM		IonQ	
	SimuQ	Qiskit	SimuQ	Qiskit
1	**0.724**	0.855	0.240	**0.114**
2	**1.365**	1.929	**0.453**	0.474
3	**2.082**	3.166	**0.518**	0.715

$R_{Z_jX_k}(\theta) = e^{-i(\theta/2)Z_jX_k}$. It then invokes Qiskit's transpiler to decompose each $R_{Z_jX_k}(\theta)$ into two CNOT gates and several single qubit gates and generates a Qiskit pulse schedule, which is displayed in Figure 4.10 and is around 1660ns long. This is around six times longer than the pulse schedule generated by SimuQ using the IBM-native AAIS.

4.5.3 Hamiltonian-oriented compilation with interaction-based gates

For some devices that lack the support of simultaneous instruction executions by native operations, we can still exploit the capability of realizing gates based on evolving interaction for various time periods. By interaction-based gates, we refer to quantum gates of form $R_H(t) = e^{-itH}$, where the time duration of the pulse shapes implementing them is strongly correlated with t. These gates are common on platforms supporting universal gates like IBM devices and IonQ devices but are not exploited in their provided compiler due to the hardness in calibration. Under the conventional circuit-oriented compilation where quantum programs are compiled to a gate set with fixed number 2-qubit gates (typically, only CNOT gates), interaction-based gates are decomposed using multiple 2-qubit gates for the convenience of calibration, like $R_{Z_jX_k}(\theta)$ gates that are decomposed using 2 CNOT gates by Qiskit and translated to a pulse schedule of long and fixed duration. Although interaction-based gates cannot be simultaneously applied on devices when overlapping sites exist, exploiting them can still significantly reduce the duration of generated pulse schedules and increase the fidelity of simulations as observed in [64, 67].

In this section, we implement the quantum approximate optimization algorithm (QAOA) in SimuQ and execute it on IBM and IonQ devices. The QAOA algorithm is a classical-quantum Hamiltonian-oriented algorithm designed to solve combinatorial problems. We omit the algo-

rithm analysis and refer interested readers to [68]. We consider the quantum simulation part of a typical case of the QAOA algorithm, where the target quantum system evolves under a length-p sequence of alternative evolution between H_1 and H_2 where

$$H_1 = Z_1 Z_N + \sum_{j=1}^{N-1} Z_j Z_{j+1}, \qquad H_2 = \sum_{j=1}^{N} X_j. \tag{4.5.3.1}$$

Here $N = 12$ is the problem size. Two pre-defined parameter lists $\{\theta_j\}_{j=1}^{p}$ and $\{\gamma_j\}_{j=1}^{p}$ of length p describe the time of each evolution segment. I.e., the j-th segment first lets the system evolve under H_1 for time θ_j and then lets the system evolve under H_2 for time γ_j. Ultimately, we measure the sites and store the results in a bit-string s. The more precisely we simulate the system, the larger the evaluation function $C(s) = |\{j : 1 \leq j \leq N, s_j \neq s_{j+1}\}|$ will be (assuming $s_{N+1} = s_1$).

When compiling the target system for the Heisenberg AAIS, $\eta_{j,j+1,ZZ}$ are executed frequently. We take an execution $\eta_{1,2,ZZ}$ for $t = 1$ as an example, which effectively realizes the gate $e^{-iZ_1Z_2}$. Qiskit decomposes it with 2 CNOT gates and single qubit rotation gates, and the generated pulse duration takes a constant 662ns independent of t, as illustrated in Figure 4.12. An alternative solution is to create Z_1X_2 interaction constructed by echoed cross-resonance pulses for a duration positively correlated to t with short pulses implementing Hadamard gates to effectively realize the Z_1Z_2 interaction, as illustrated in Figure 4.11. The pulse schedule is around $(200t + 130)$ns long to execute $\{\!|\eta_{j,j_1,ZZ}|\!\}(1)$ for time t. When $t = 1$, it is 324ns long, 51% shorter than the Qiskit compiler. Interaction-based gates are especially beneficial when a program requires many short instruction executions, like compiled simulations with a large Trotterization number.

We then compile and execute the QAOA system simulation to IBM and IonQ devices for cases $p = 1, 2, 3$. Similarly, we reproduce this problem in Qiskit and compile it with the Qiskit compiler (CNOT-based decomposition is applied). On IBM devices, for $p = 3$, the pulse schedule generated by SimuQ is 3.35ms long. In contrast, the one generated by Qiskit is 7.48ms long, which is more than two times longer. On average, SimuQ generates pulse schedules 59% percent shorter than Qiskit. We then execute the generated programs on IBM devices and IonQ devices. The differences of the evaluation function $C(s)$ measured on devices and ground truth values are present in Table 4.2. We observe that, on average, the pulse schedules generated by SimuQ reduce errors (the difference to ideal results) by 34% on IBM devices and 28% on IonQ devices for $p = 3$ compared to pulse schedules from Qiskit. The advantage is less significant for shallow cases where $p = 1, 2$ because of state preparation and measurement errors on real devices [69]. These experiments demonstrate the advantage and the necessity of Hamiltonian-oriented compilation using interaction-based gates on devices not supporting simultaneous instruction executions.

4.5.4 Benchmarking quantum simulation compilation

To illustrate SimuQ's capability of dealing with various quantum simulation problems, we craft a small benchmark containing models collected from multiple domains like condensed matter physics, high-energy physics, particle physics, and optimization. The diversity of the cases in this benchmark of different topologies, time dependency, and system sizes exhibits our compiler's feasibility and efficiency in dealing with significant simulation problems.

We present the benchmark in Table 4.3. We report each quantum system's number of sites and lines of code to implement them with HML in SimuQ. Most systems can be programmed

Table 4.3: A benchmark of quantum simulation problems. We program and compile the models in SimuQ to obtain pulse schedules for QuEra and IBM devices and quantum circuits for IonQ devices. We record the compilation time (comp. time), the pulse duration (P.D.), and the 2-qubit gate count for the generated circuits. No sol. represents cases where the SimuQ compiler reports no solution because of hardware constraints, such as limited interactions of QuEra devices and machine topology for IBM devices. Time out is reported when the compilation takes more than an hour, which happens in the search for a 64-qubit cycle on IBM devices.

System name	**LoC**	**# of sites**	**QuEra**	**IBM**			**IonQ**	
			Comp. time (s)	**Comp. time** (s)	**P.D.** (μs) **SimuQ**	**P.D.** (μs) **Qiskit**	**Comp. time** (s)	**# of 2q-gate**
ising_chain	13	6	0.177	0.224	2.06	8.69	0.155	20
		32	39.3	54.6	3.24	39.2	47.2	124
		64	663	257	3.15	81.2	680	252
		96	2298	1086	3.26	450	3568	380
ising_cycle	13	6	0.585	No sol.			0.13	24
		12	3.47	1.49	2.05	37.8	1.37	48
		32	114	483	3.35	144	53.8	128
		64	3454	Time out			907	256
heis_chain	15	32	No sol.	143	10.1	119	138	372
qaoa_cycle	19	12	No sol.	0.503	0.83	37.6	1.5	36
qhd	16	16	No sol.	No sol.			66.3	480
mis_chain	22	12	5.45	19.1	18.9	94	25.2	440
		24	53.1	328	18.9	162	278	920
mis_grid	29	16	28.4	No sol.			85.4	960
		25	141	No sol.			489	1600
kitaev	13	18	4.67	15.6	2.12	21.2	8.74	68
schwinger	18	10	No sol.	No sol.			1.09	28
o3nlσm	19	30	No sol.	No sol.			77.7	588

within 20 lines, showing the user-friendliness of programming quantum systems in SimuQ.

For each target quantum system, we compile it on the platforms supported by SimuQ using their most capable devices in the possible future. The compilation time is averaged over 5 runs on a laptop with Intel Core i7-8705G CPU. SimuQ compiler reports no solution (No sol.) in several cases due to complicated interactions beyond the hardware capability of QuEra devices and the limited connectivity of IBM devices. Limited connectivity on large IBM devices also complicates the site layout search, making a case exceed a pre-set compilation time limit of 3600 seconds, which is marked as a time out.

Pulse schedule duration for IBM devices using SimuQ and Qiskit to compile is reported. On average, Qiskit's default compilation passes generate 29.3 times longer pulse schedules than the SimuQ compiler over cases successfully compiled. We also report the number of partially entangling Mølmer-Sørenson gates when compiling on IonQ's devices to indicate the total duration.

4.6 Follow-up works

SimuQ serves as a path-finding platform for applications of Hamiltonian-oriented quantum computing. Numerous follow-up projects of SimuQ are in progress upon the completion of this dissertation, like realizing Hamiltonian-based algorithms on IonQ devices and evaluating the performance of dynamical decoupling sequences in quantum simulation. A particularly interesting one is the development of QHDOPT[70], which we briefly discuss here.

QHDOPT. The QHD-based OPTimization (QHDOPT) package is a software solution for non-linear and non-convex optimization problems, utilizing the design of Hamiltonian-oriented quan-

tum computing. A Hamiltonian-based algorithm, Quantum Hamiltonian Descent (QHD) [71], is implemented in SimuQ and wrapped for domain users.

The package targets problems of form

$$\min_x f(x_1, \ldots, x_n) = \underbrace{\sum_{i=1}^{n} g_i(x_i)}_{\text{univariate part}} + \underbrace{\sum_{j=1}^{m} p_j(x_{k_j}) q_j(x_{\ell_j})}_{\text{bivariate part}}, \tag{4.6.0.1a}$$

$$s.t.\ L_i \leq x_i \leq U_i, \forall i \in \{1, \ldots, n\}, \tag{4.6.0.1b}$$

where $x_1, \ldots, x_n$ are n variables subject to the box constraint $x_i \in [L_i, U_i] \subset \mathbb{R}$ for each $i = 1, \ldots, n$, and the indices $k_j, \ell_j \in \{1, \ldots, n\}$ and $k_j \neq \ell_j$ for each $j = 1, \ldots, m$. The functions $g_i(x_i)$, $p_j(x_{k_j})$, and $q_j(x_{\ell_j})$ are real univariate differentiable functions defined on $\mathbb{R}$. Note that the univariate part in (4.6.0.1a) has at most n terms because we can always combine separate univariate functions of a fixed variable x_i into a single one. However, there is no upper bound for the integer m (i.e., the number of bivariate terms).

To input the objective function and constraints, QHDOPT provides an interface through SymPy [72] and also a specific interface for box-constrained quadratic programming via matrices. QHDOPT takes the problem instance to construct a Hamiltonian evolution according to QHD algorithm, then programs it in SimuQ and deploy it to D-Wave devices, IonQ devices, and classical simulators. We evaluate our package on a small benchmark of nonconvex optimization problems, and obtained more than 10 times speedup compared to state-of-the-art classical solvers like Ipopt [73] and Gurobi [74]. More details of the evaluation are in [70].

Chapter 5: Automatic Differentiation of Parameterized Quantum Systems

Chapter summary

One of significant software tools in modern classical computing is automatic differentiation of parameterized systems like neural networks. In this chapter, we develop the theory of automatic differentiation for parameterized quantum evolution, whose programming can be dealt with using SimuQ (see Chapter 4).

We formulate the first differentiable analog quantum computing framework with a specific parameterization design at the analog signal (pulse) level to better exploit near-term quantum devices via variational methods. We further propose a scalable approach to estimate the gradients of quantum dynamics using a forward pass with Monte Carlo sampling, which leads to a quantum stochastic gradient descent algorithm for scalable gradient-based training in our framework. Applying our framework to quantum optimization and control, we observe a significant advantage of differentiable analog quantum computing against SOTAs based on parameterized digital quantum circuits by *orders of magnitude*.

The theory studied in this chapter serves as the foundation of building a software stack integrating automatic differentiation for quantum systems, which lead to diverse applications in quantum chemistry and quantum control.

5.1 Introduction

Quantum computing has promised unprecedented improvement in our computational ability to tackle classically intractable problems. Despite the rapid development of quantum hardware [75, 76], near-term quantum computers are still likely to have very restricted hardware resources, where the limited number of "qubits" and non-negligible machine noises would impede the implementation of large-scale quantum algorithms. Recent research results in both computer science [77] and physics [78] suggest a promising approach to designing resource-efficient *Noisy Intermediate-Scale Quantum (NISQ)* [34] applications by breaking quantum circuit abstractions and directly designing applications at the pulse-level control of quantum machines. The benefits of this Hamiltonian-oriented approach have been witnessed in the history of classical analog computing that predates digital computing due to relaxed hardware requirement and plays an important role in domain applications such as simulation.

One leading algorithmic paradigm on NISQ computers is the *Variational Quantum Algorithm (VQA)* with a few prominent examples like the Variational Quantum Eignensolver (VQE) [79], quantum approximate optimization algorithm (QAOA) [68], and more in [80]. Specifically, VQA uses *parametrized quantum models* to characterize loss functions, in particular those from quantum physics, which are potentially intractable for classical computing [81]. These parameters will then be optimized, usually through gradient-based approaches, to minimize the given loss function. Conventionally, parameterized quantum models are typically quantum circuits where each gate is parameterized by classical variables. Moreover, auto-differentiation techniques have been recently developed (e.g., [82, 83, 84]) for a scalable quantum gradient calculation on parameterized quantum circuits. We henceforth refer this existing framework as *differentiable digital*

	Diff. Quantum	Neural ODE	Diff. Physics
$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}(t) = ?$	$-iH(\mathbf{v}, t)\mathbf{x}(t)$	$f(\mathbf{x}(t), \mathbf{v})$	$f(\mathbf{x}(t), \mathbf{v})$
Parametrization	Basis Function	Neural Network	Dynamics Equation
Forward	Time Evolution	Forward Pass	Time Integration
Backward	MC Integration	Autodiff	Auto/Manual Diff.
Device	Quantum	Classical	Classical

Table 5.1: Machine learning in different dynamical systems.

quantum computing.

Although parameterized quantum circuits are designed for NISQ applications, implementing (digital) quantum circuits still incurs non-negligible overheads, which significantly restricts the size of parametrized circuits that can be executed faithfully on NISQ machines. Moreover, the current parameterization in VQAs is also largely restricted by available parameterized gates and how they compose circuits, which in turn limits the expressive power of VQAs. A natural alternative to the current digital parameterization is to perform VQA directly on *parametrized analog signals (pulses)*, either on digital quantum machines with pulse-level controls, *or* on general analog quantum hardware. Indeed, parameterized analog pulses have the potential for more efficient NISQ implementation and better expressiveness as suggested in a recent perspective paper [78], which could be a more favorable parameterization for NISQ applications even when digital quantum gates are available.

However, there is not yet a systematic study of the analog parameterization for VQAs, its scalable training, and its quantitative benefits in achieving high-quality solutions in VQA applications.

Contributions. We conduct the first systematic study of the parameterization and scalable training of analog VQAs, which we call *differentiable analog quantum computing*. We also leverage our scalable training to demonstrate the quantitative benefits of the analog parameterization for

specific VQA applications.

Specifically, we formulate the general differentiable analog quantum computing as a control problem, $\min_{\mathbf{v}} \mathcal{L}(\mathbf{v})$, where the loss function $\mathcal{L}(\mathbf{v})$ is calculated from the $\mathbf{v}$-parametrized quantum state $\mathbf{x}(T;\mathbf{v})$ generated by quantum machines at the evolution time $t = T$. Intuitively, this control problem is no different from any classical ones except that the evolution of the quantum state in $[0, T]$ is governed by the Schrödinger equation

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}(t) = -iH(\mathbf{v}, t)\mathbf{x}(t), \tag{5.1.0.1}$$

where the Hamiltonian operator $H(\mathbf{v}, t)$ can be much more flexibly parameterized in $\mathbf{v}$ compared with parameterized quantum circuits (detailed in Section 5.2.2), and i is the imaginary unit.

We also develop a scalable Monte Carlo integration technique of computing quantum-related gradients from the loss function $\mathcal{L}(\mathbf{v})$. A well-known difficulty in computing quantum-related gradients by classical means is the exponential cost associated with classical simulation of the quantum system. Thus, any scalable solution must leverage quantum machines to compute the gradients for themselves. Existing "auto-differentiation" techniques for parameterized quantum circuits (e.g., [82, 83, 84]) are designed for discrete-time evolution and the basic parameterized units (i.e., gates) are described by finite-dimensional matrices. Differential analog quantum computing exploits parameterization of continuous-time evolution and $H(\mathbf{v}, t)$ refers to a parameterized model from a functional space. Our Monte Carlo integration technique is designed to bridge this technical gap, which is later integrated with the stochastic gradient descent (SGD) for the entire framework for fast convergence and robustness against empirical noise. We further illustrate that our quantum stochastic gradient descent could still work with approximate

descriptions of $H(\mathbf{v}, t)$ for domain applications.

An analogy could be drawn between our approach and classical deep learning, as summarized in Table 5.1. *Neural ODEs* [85, 86] view the structure of ResNet [87], $\mathbf{x}_{n+1} = \mathbf{x}_n + f(\mathbf{x}_n, \theta)$, as the solving of an ordinary differential equation,

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}(t) = f(\mathbf{x}(t), \mathbf{v}), \tag{5.1.0.2}$$

with $f(\cdot)$ as the network layer, $\mathbf{v}$ the network parameter, and $\mathbf{x}(t)$ the hidden state. This formula is similar to our system in (5.1.0.1), although we adopt a different parametrization than neural networks. Similarly, *differentiable physics* (e.g., [88]), which incorporates physical simulation into learning process, leverages existing numerical solvers and the autodiff functionality of deep learning with backpropagation to compute gradients of a physical or dynamical system, then integrates them into a neural network. This approach has proven to accelerate learning and generalization.

Differentiable analog quantum computing can be deemed as a special form of differentiable physics at the quantum scale. Similar to the promise of differentiable physics or neural ODEs, we have also observed the advantageous performance of differentiable analog quantum computing against the conventional parameterized quantum circuits by orders of magnitude in major VQA applications like optimization (Section 5.3) and control (Section 5.4). Our auto-differentiation technique in quantum is however less efficient than classical ones that leverage the chain rule and the forward/backward propagation. This is due to the quantum no-cloning theorem [89] which prevents us from storing any arbitrary immediate state during quantum computation. So, we develop a sampling technique to compute gradients. To sum up, the key contributions of this

work include:

- A formulation of differentiable, resource-efficient *analog* quantum computing framework (Sec. 5.2),
- A scalable technique of computing gradients by Monte Carlo sampling with SGD (Sec. 5.2.3),
- Formal analysis on correctness, efficiency, and robustness that showcases *exponential reduction of computational complexity* and bounded errors with approximate machine description (Sec. 5.2.4-5.2.6),
- Applications of our framework on quantum optimization (Sec. 5.3) and control (Sec. 5.4), with demonstrated advantages by *orders of magnitude* against parameterized quantum circuits. Our code is available here: `https://github.com/YilingQiao/diffquantum`

Related work.

Learning for dynamics and control. Dynamical systems have widely been used to interpret and improve the design of machine learning algorithms. Compared to traditional discrete layers, Neural ODEs [90] demonstrate that continuous modeling of neural network can better learn the continuous structures [91, 92] with infinite depth [93] and constant memory cost [94]. Besides neural networks, differentiable physics [95] also computes analytic gradients of classical dynamical systems like rigid body [96, 97], articulated body [98, 99, 100], soft body [101, 102, 103], and fluids [104, 105, 106, 107]. Those methods have made great success on design [108], control [109] and system identification [110] tasks in the macroworld. Our framework, as the first differentiable dynamics for quantum computing, could have tremendous potential in chemistry and physics applications.

Quantum machine learning & optimal control. Quantum machine learning is a fast-developing emerging field (e.g., see the survey [111]) where variational quantum algorithms (VQAs) (e.g., see the survey [80] are one of the most promising candidates for NISQ applications. Quantum optimal control (succinctly, quantum control) aims to achieve a desired response from the quantum system by controlling the system parameters [112, 113]. Quantum control theory has empowered the growth of quantum technologies and found applications in several fields, ranging from molecular chemistry to quantum computing [114, 115]. The connection between quantum control theory and VQAs has been discussed recently [78, 116]. Several optimization algorithms have been developed to solve quantum control problems, including the Krotov method [117], monotonically convergence algorithms [118], non-iterative algorithms [119], the *GRadient Ascent Pulse Engineering (GRAPE)* algorithm [120], the *Chopped RAndom-Basis (CRAB)* algorithm [121], etc. The development of quantum computing also opens up the possibility of solving quantum control problems with quantum computers [122, 123]. These proposals take the approach of hybridizing quantum simulations with classical optimization routines. Nevertheless, they either do not support gradient-based methods or compute the gradients in a non-scalable way (e.g., via classical simulation), which significantly limits their performance especially comparing to ours. Existing pulse-level variational algorithms [116, 124, 125, 126] do not discuss their direct application to quantum analog machines and can not compute gradients analytically on quantum machines, while our method resolves this problem.

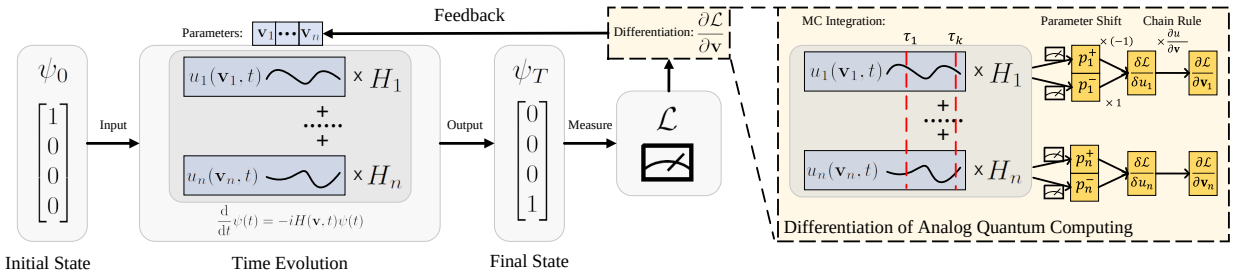


Figure 5.1: The workflow of differentiable analog quantum computing is presented. In this 2-qubit example, the system starts with an initial ground state ψ_0 of dimension $4 = 2^2$ and evolves through the time following the Schrödinger equation (5.1.0.1). We control the dynamics of this quantum system by specifying the time-dependent Hamiltonian $H(\mathbf{v}, t)$, parameterized by trainable variables $\mathbf{v}$. At the end of the process, we measure the system and get a real-valued loss value, $\mathcal{L}$. The derivatives are computed as in the right box. Quantum computers cannot store computational graphs, so we propose to sample a time t in the forward simulation and apply the parameter shift rule to compute the gradients. The derivatives are then used in the feedback loop to update $\mathbf{v}$ optimizing $\mathcal{L}$.

5.2 Differentiation of parameterized quantum systems

Similar to classical dynamical systems, quantum systems also have states, governing equations, and observations, so there naturally exist plenty of optimization [127], control [128], and learning [111] problems for quantum computing. Inspired by the success of gradient-based approach in the classical domain, we propose a differentiable framework to compute the gradients of parametrized analog control signals on quantum computers, based on a "*forward simulation with stochastic sampling*" (see the workflow in Figure 5.1).

5.2.1 Quantum preliminaries

A *qubit* (or *quantum bit*) is the analogue of a classical bit in quantum computation. It is a two-level quantum-mechanical system described by vectors in the Hilbert space $\mathbb{C}^2$. We use Dirac notation $|\psi\rangle$ to denote quantum states (i.e., unit vectors) ψ in $\mathcal{H}$. For example, the classical

"0" and "1" are represented by $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. One qubit states could be in any linear combination of $|0\rangle$ and $|1\rangle$, which is called *superposition*. An n-qubit state is a unit vector in the Kronecker tensor product $\otimes$ of n single-qubit Hilbert spaces, i.e., $\mathcal{H} = \otimes_{i=1}^{n} \mathbb{C}^2 \cong \mathbb{C}^{2^n}$, whose dimension is exponential in n. For an n by m matrix A and a p by q matrix B, their Kronecker product is an np by mq matrix where $(A \otimes B)_{pr+u,qs+v} = A_{r,s} B_{u,v}$. The **complex conjugate transpose** of $|\psi\rangle$ is denoted as $\langle\psi| = |\psi\rangle^\dagger$ ($\dagger$ is the Hermitian conjugate). Therefore, the **inner product** of ϕ and ψ could be written as $\langle\phi\rangle\,\psi$.

The time evolution of quantum states under Schödinger equation is specified by the (time-dependent) Hermitian matrix $H(t)$ over the corresponding Hilbert space, known as the *Hamiltonian* of the quantum system. Typical single-qubit Hamiltonians include the famous *Pauli matrices*:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{5.2.1.1}$$

Similarly, a multi-qubit Hamiltonian can be built from the Pauli group consisting of tensor products of Pauli matrices. Conventionally we write X_j (Y_j, Z_j) for a multi-qubit Hamiltonian to indicate the tensor product of multiple identity matrices while the j-th operand is X (Y, Z), which represents operations on the j-th subsystem.

Quantum measurement refers to the procedure of extracting classical information from quantum systems. It is characterized by an Hermitian matrix M called the *observable*. Measuring a quantum state $|\psi\rangle$ with observable M is modelled as a random variable whose expectation value is $\langle\psi|\, M\, |\psi\rangle$.

5.2.2 Problem formulations

Most computational tasks in quantum simulation and control, like finding the ground state of a physics system, can be formulated as the following optimization problem. Given a quantum observable M and an initial state $|\psi(0)\rangle = |\psi_0\rangle$, we seek for a parameter vector $\mathbf{v}$ by minimizing the loss function

$$\mathcal{L} = \langle\psi(T)| M |\psi(T)\rangle , \tag{5.2.2.1}$$

where the evolution of $|\psi(t)\rangle$ from $t = 0$ to $t = T$ subject to the Schrödinger equation (5.1.0.1). Here, $H(\mathbf{v}, t)$ is a Hamiltonian parametrized by $\mathbf{v}$ with form

$$H(\mathbf{v}, t) = H_c + \sum\nolimits_{j=1}^{m} u_j(\mathbf{v}, t) H_j, \tag{5.2.2.2}$$

where m is the number of control pulses, H_c is a time-independent Hamiltonian (e.g. Pauli matrices), H_j are tensor products of Pauli matrices, and the range of $u_j(\mathbf{v}, t)$ is $\mathbb{R}$. We also require $u_j(\mathbf{v}, t)$ to be differentiable with respect to $\mathbf{v}$ for any $t \in [0, T]$. The loss function $\mathcal{L}$ is hence *differentiable*. We can loosen the restriction of H_j to general time-independent Hamiltonians if we have powerful enough quantum simulators. We keep it for the convenience of presenting our algorithm.

With specific M and $|\psi_0\rangle$, optimization problem $\min_{\mathbf{v}} \mathcal{L}$ with post-processing can encode many essential classical and quantum problems. For example, any classical optimization of $f(x)$ over n-bit integers x can be formulated as $M = \sum_x f(x) |x\rangle \langle x|$. More examples like Max-Cut problem and quantum state preparation are discussed in details in Sec. 5.3, 5.4.

Abstract Quantum Analog Machines (AQAMs). We propose AQAMs as our computational

model. An AQAM optimizing the above loss function should be capable of consecutively: (1) evolving under $H(\mathbf{v}, t)$ for any $\mathbf{v}$ and time interval $[t_1, t_2] \subset [0, T]$; (2) evolving under constant Hamiltonian H_j for time duration τ (effectively applying unitary transformation $e^{-iH_j\tau}$); (3) preparing $|\psi_0\rangle$; (4) measuring with observable M. For realistic quantum devices, we need to design AQAMs accordingly.

Remark 5.2.1. *A trivial AQAM directly employs the Hamiltonian of a quantum device as $H(\mathbf{v}, t)$, parameterized by tunable pulses. In most realistic quantum devices, multi-qubit interactions are not tunable and weak compared to tunable single-qubit Hamiltonians. Thus H_j can be simulated with high fidelity. Our method is robust against imprecise simulations of $H(\mathbf{v}, t)$ and H_j (discussed in Section 5.2.6). Hence the trivial AQAMs are usually suitable for realistic quantum devices.*

Our formulation of the problem via analog quantum computing is a generalization of the formulation via parameterized circuits. For example, a series of parameterized Pauli rotation gates $R_{P_j}(\theta_j) = \exp\{-i(\theta_j/2)P_j\}$ can be realized by $H(\mathbf{v}, t) = \sum_j \mathbf{v}_j \mathbf{1}_j(t) P_j$ with valuation $\mathbf{v}_j = \theta_j/2$, where $\mathbf{1}_j$ is the indicator function of $[j, j+1]$. However, simulating analog quantum computing via quantum circuits requires much longer time on nowadays devices [78], hence is unrealistic. So direct analog quantum computing can *exploit near-term quantum devices much better* than quantum circuits.

5.2.3 Quantum stochastic gradient descent

Our quantum SGD scheme for computing gradients on AQAMs is illustrated in this section, with its correctness, efficiency, and robustness discussed in the following sections.

Algorithm 2 Estimating gradients on an AQAM

Classical inputs: $b_{\text{int}}, b_{\text{obs}}$ (batch sizes), m (number of control pulses), u_j (parametrized pulses), T (evolution duration), $\mathbf{v}$ (parameters)
Quantum inputs: $\mathcal{E}_{|\psi_0\rangle}$ (preparation of initial state), H (parametrized system Hamiltonian), H_j (pulse-controllable Hamiltonian), $\mathcal{E}_M$ (measurement procedure with observable M)
Output: a gradient estimation of $\mathcal{L}$ to $\mathbf{v}$

for $k \in \{1, ..., b_{\text{int}}\}$ **do**
  Draw $\tau \sim \text{Uniform}(0, T)$
  for $j \in \{1, ..., m\}, s \in \{-1, +1\}$ **do**
    for $l \in \{1, ..., b_{\text{obs}}\}$ **do**
      Prepare $|\psi_0\rangle$ via $\mathcal{E}_{|\psi_0\rangle}$
      Evolve under $H(\mathbf{v}, t)$ for time $[0, \tau]$
      Evolve under H_j for duration $(1 + \frac{3}{4}s)\pi$
      Evolve under $H(\mathbf{v}, t)$ for time $[\tau, T]$
      $x_l \leftarrow$ observation sample from $\mathcal{E}_M$
    $\tilde{p}_j^{\text{sign}(s)} \leftarrow \frac{1}{b_{\text{obs}}} \sum_{l=1}^{b_{\text{obs}}} x_l$
  $\tilde{f}_k \leftarrow \sum_{j=1}^{m} \frac{\partial u_j}{\partial \mathbf{v}}(\mathbf{v}, \tau)(\tilde{p}_j^- - \tilde{p}_j^+)$
$\widetilde{\frac{\partial \mathcal{L}}{\partial \mathbf{v}}} \leftarrow \frac{T}{b_{\text{int}}} \sum_{k=1}^{b_{\text{int}}} \tilde{f}_k$

We borrow the idea of mini-batches from classical SGD to deal with the gradients, and set two layers of mini-batches: (1) an integration mini-batch with size b_{int}; (2) an observation mini-batch with size b_{obs}. The integration mini-batch updates parameters according to the estimation of derivatives on the sampled time. The observation mini-batch repeats experiments to generate more precise measurement results. The scheme is displayed in Algorithm 2.

The forward and backward propagation of our SGD scheme is depicted in Figure 5.1. Notice that the inner loop is the only procedure on quantum machine. The difference of this procedure compared with the estimation of loss function $\mathcal{L}$ is the inserted evolution under H_j, which is beneficial in the error analysis in Section 5.2.6.

With the estimation of gradients, various optimizers designed for classical stochastic gradient descent are suitable to optimize the objective function. For example, Adam [129] is used in our experiments.

5.2.4 Correctness of gradient estimation

We show that Algorithm 2 generates an unbiased estimation of the gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{v}}$, and hence establishes its correctness.

Theorem 5.2.1. *The derivative of $\mathcal{L}$ to parameters $\mathbf{v}$ is*

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}} = \int_0^T \mathrm{d}\tau \sum\nolimits_{j=1}^{m} \frac{\partial u_j}{\partial \mathbf{v}}(\mathbf{v}, \tau) \left(p_j^-(\tau) - p_j^+(\tau)\right). \tag{5.2.4.1}$$

Here, $p_j^{\pm}(t) = \langle \psi_0 | U_{\mathbf{v}}^{\dagger}(t,0) e^{iH_j(1\pm\frac{3}{4})\pi} U_{\mathbf{v}}^{\dagger}(T,t) M U_{\mathbf{v}}(T,t) e^{-iH_j(1\pm\frac{3}{4})\pi} U_{\mathbf{v}}(t,0) | \psi_0 \rangle$, where $U_{\mathbf{v}}(t_2,t_1)$ denotes the time evolution operator for time interval $[t_1,t_2]$ under Hamiltonian $H(\mathbf{v},t)$.

Proof. We first show that the derivative of the time evolution operator is

$$\frac{\partial U_{\mathbf{v}}(t_2,t_1)}{\partial \mathbf{v}} = -i \int_{t_1}^{t_2} \mathrm{d}\tau\, U_{\mathbf{v}}(t_2,\tau) \frac{\partial H(\mathbf{v},\tau)}{\partial \mathbf{v}} U_{\mathbf{v}}(\tau,t_1) \tag{5.2.4.2}$$

Let $U_{\mathbf{v}}(t_2,t_1)$ be as defined in Theorem 5.2.1. We can re-write (5.1.0.1) in the form of time evolution operator,

$$i\frac{\partial U_{\mathbf{v}}(t,0)}{\partial t} = H(\mathbf{v},t)U(t,0). \tag{5.2.4.3}$$

It follows that

$$\begin{aligned}
& i\left(U_{\mathbf{u}}(t,0) - U_{\mathbf{v}}(t,0)\right)' \\
=& H(\mathbf{u},t)U_{\mathbf{u}}(t,0) - H(\mathbf{v},t)U_{\mathbf{v}}(t,0) \\
=& [H(\mathbf{u},t) - H(\mathbf{v},t)]U_{\mathbf{u}}(t,0) + H(\mathbf{v},t)(U_{\mathbf{u}} - U_{\mathbf{v}})(t,0).
\end{aligned} \tag{5.2.4.4}$$

By the variation-of-parameters formula [60], we have

$$U_{\mathbf{u}}(t,0) - U_{\mathbf{v}}(t,0) = -i \int_0^t \mathrm{d}\tau\, U_{\mathbf{v}}(t,\tau)[H(\mathbf{u},\tau) - H(\mathbf{v},\tau)]U_{\mathbf{u}}(\tau,0). \tag{5.2.4.5}$$

Now, if $\mathbf{u} = \mathbf{v} + h$, we have $H(\mathbf{u},t) = H(\mathbf{v},t) + h\frac{\partial H(\mathbf{v},t)}{\partial \mathbf{v}} + O(h^2)$. Therefore, for each $t \geq 0$, we have

$$\frac{U_{\mathbf{u}}(t,0) - U_{\mathbf{v}}(t,0)}{h} = -i \int_0^t \mathrm{d}\tau\, U_{\mathbf{v}}(t,\tau)\frac{\partial H(\mathbf{v},\tau)}{\partial \mathbf{v}} U_{\mathbf{u}}(\tau,0) + O(h), \tag{5.2.4.6}$$

which implies the desired result (5.2.4.2) by taking $h \to 0$.

This proofs (5.2.4.2). We now take the derivative to $\mathcal{L}$. Let $\mathcal{L}$ be defined as (5.2.2.1), and $H(\mathbf{v},t) = \sum_j f_j(\mathbf{v},t)H_j$. Then

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{v}} &= \langle\psi(0)|\, U^\dagger(T,0) M \frac{\partial U(T,0)}{\partial \mathbf{v}} \,|\psi(0)\rangle + h.c. \\
&= \Big(-i \int_0^T \mathrm{d}\tau \sum_j \frac{\partial f_j(\mathbf{v},\tau)}{\partial \mathbf{v}} \langle\psi(0)|\, U^\dagger(\tau,0) U^\dagger(T,\tau) M U(T,\tau) H_j U(\tau,0)\,|\psi(0)\rangle \Big) + h.c. \qquad (5.2.4.7) \\
&= \Big(-i \int_0^T \mathrm{d}\tau \sum_{j=1}^m \frac{\partial f_j(\mathbf{v},\tau)}{\partial \mathbf{v}} \langle\psi(\tau)|\, M_\tau H_j\,|\psi(\tau)\rangle \Big) + h.c., \qquad (5.2.4.8)
\end{aligned}$$

where $|\psi(\tau)\rangle = U(\tau,0)\,|\psi(0)\rangle$ is the state evolving under $H(\mathbf{v},t)$ to time τ starting from $|\psi(0)\rangle$, and $M_\tau = U^\dagger(T,\tau)MU(T,\tau)$. Here $h.c.$ means the Hermitian conjugate of the first half of the expression (this is a common abbreviation among physics, resulting in a Hermitian matrix). □

One can interpret (5.2.4.1) as a direct application of the chain rule over functional derivatives $\frac{\delta \mathcal{L}}{\delta u_j}$ and partial derivatives $\frac{\partial u_j}{\partial \mathbf{v}}$, since $\frac{\delta \mathcal{L}}{\delta u_j}(\mathbf{v},t) = p_j^-(t) - p_j^+(t)$ by the parameter shift rule

[82, 83], which is a technique for evaluating commutators of Hermitian by quantum processes.

Algorithm 2 estimates the integral in (5.2.4.1) via Monte Carlo integration (MCI) technique. We show the MCI generates an unbiased estimation of $\frac{\partial \mathcal{L}}{\partial \mathbf{v}}$ and converges of rate $O(b_{\text{int}}^{-\frac{1}{2}})$. We require that the derivatives of $u(\mathbf{v}, t)$ are bounded: $\forall t \in [0, T]$, $\left|\frac{\partial u_j(\mathbf{v},t)}{\partial \mathbf{v}}\right| \leq D$.

The integration mini-batch draws time samplings $\tau \sim \text{Uniform}(0, T)$, and evaluates

$$f(\tau) = \sum_{j=1}^{m} \frac{\partial u_j}{\partial \mathbf{v}}(\mathbf{v}, \tau) \left(p_j^-(\tau) - p_j^+(\tau)\right). \tag{5.2.4.9}$$

Then (5.2.4.1) turns to $\frac{\partial \mathcal{L}}{\partial \mathbf{v}} = \int_0^T f(\tau) \mathrm{d}\tau$. By MCI, the average value of $T \cdot f(\tau)$ in the integration mini-batch is an unbiased estimation of $\frac{\partial \mathcal{L}}{\partial \mathbf{v}}$. By applying Popocivius's inequality and the Cauchy-Schwarz inequality, we obtain the following variance bound, which guarantees a low error of MCI.

Proposition 5.2.2. *The variance of $f(\tau)$ is finite. Specifically,*

$$\mathit{Var}[f(\tau)] \leq 4m^2 \|M\|^2 D^2. \tag{5.2.4.10}$$

Other numerical integral methods are also applicable here for different conditions on u_j and H_j and may have better convergence rates than MCI. We present it here because MCI has good generalization and simplicity. We also remark that similar ideas developed in this section have also appeared in [130] in the circuit model, whereas we developed everything in the analog quantum computing model.

Overall, the forward and backward propagation of differentiation of $\mathcal{L}$ is depicted in Figure 5.1. Since the parameterization of u_j is arbitrary, a typical treatment is using a Fourier basis

or a Legendre basis as the support of the parameterization. Neural networks are also suitable for pulse generation, whose gradients are easy to compute via backpropagation.

5.2.5 Scalability analysis

The resource consumption of our classical-quantum hybrid approach has two aspects: the classical and the quantum sides. The classical computation, as described in Algorithm 2, has $O(b_{\text{int}} b_{\text{obs}} m)$ numerical calculations. On the quantum side, the sampling process assessing the loss function and its derivatives takes $O(T)$ time each. The total running time on a quantum computer then is $O(b_{\text{int}} b_{\text{obs}} m T)$. Almost on every architecture of quantum devices, the number of control signals m is *at most quadratic in the number of qubits* n (see some survey papers [28, 29, 131, 132]), showing excellent scalability of our approach. We exhibit the scalability of our method for up to 11 qubits in numerical experiments in Section 5.3.2. Our approach could in principle be readily applied to the most advanced existing quantum systems (e.g., [133] with $n \sim 60$). This is in sharp contrast to the case of GRAPE and CRAB algorithms, which rely on classical simulation of quantum systems with an exponential cost in n. For instance, the associated classical computation cost for $n \sim 60$ (i.e. at least 2^{60}) is prohibitively high, almost reaching the limit of supercomputers today. *Our approach makes it feasible with only* $O(n^2)$ *complexity*.

5.2.6 Robustness analysis

In this section, we analyze systematic errors of gradient estimation produced of Algorithm 2. Our goal is to optimize the loss function assessed on a realistic and noisy quantum

machine, whose capabilities of evolving under $H(\mathbf{v}, t)$ and H_j are imperfect.

As a concrete example, when using the trivial AQAMs in Remark 5.2.1, the actual Hamiltonian of the quantum device may deviate from the description $H(\mathbf{v}, t)$ in our understanding, and the simulation of H_j may be imprecise due to weak non-tunable terms in H_c. We show that our algorithm well approximates the gradient of loss function of the actual devices.

One major advantage of Algorithm 2 is that even though we have imperfect realization of Hamiltonian $H(\mathbf{v}, t)$ built in the AQAM, the quantum part is executed on the actual quantum machine following the accurate Hamiltonian $\widehat{H}(\mathbf{v}, t)$. As a result, the output of our algorithm well approximates the gradient for the actual quantum machine.

Lemma 5.2.3. *Let $\widehat{\frac{\partial \mathcal{L}}{\partial \mathbf{v}}}$ denote the accurate gradient of the loss function of the quantum machine, $\frac{\partial \mathcal{L}}{\partial \mathbf{v}}$ denote the estimated gradient via Algorithm 2, and $\|\cdot\|$ represents the matrix spectral norm [134], then $\left|\frac{\partial \mathcal{L}}{\partial \mathbf{v}} - \widehat{\frac{\partial \mathcal{L}}{\partial \mathbf{v}}}\right| \leq 2\|M\| T \max_{\tau \in [0,T]} \left\|\frac{\partial H}{\partial \mathbf{v}}(\mathbf{v}, \tau) - \frac{\partial \widehat{H}}{\partial \mathbf{v}}(\mathbf{v}, \tau)\right\|$.*

Proof. We let $\widehat{\frac{\partial \mathcal{L}}{\partial \mathbf{v}}}$ denote the accurate gradient of the loss function of the quantum machine, and $\frac{\partial \mathcal{L}}{\partial \mathbf{v}}$ denote the estimated gradient via Algorithm 2. Their analytical expressions are

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}} = -i \int_0^T \mathrm{d}\tau \, \langle \psi_0 | \, \widehat{U}_{\mathbf{v}}^\dagger(T, 0) M \widehat{U}_{\mathbf{v}}(T, \tau) \frac{\partial H}{\partial \mathbf{v}} \widehat{U}(\tau, 0) \, | \psi_0 \rangle + h.c., \tag{5.2.6.1}$$

$$\widehat{\frac{\partial \mathcal{L}}{\partial \mathbf{v}}} = -i \int_0^T \mathrm{d}\tau \, \langle \psi_0 | \, \widehat{U}_{\mathbf{v}}^\dagger(T, 0) M \widehat{U}_{\mathbf{v}}(T, \tau) \frac{\partial \widehat{H}}{\partial \mathbf{v}} \widehat{U}(\tau, 0) \, | \psi_0 \rangle + h.c., \tag{5.2.6.2}$$

where $h.c.$ means the Hermitian conjugate of the first half of the expression, and $\widehat{U}_{\mathbf{v}}(t_2, t_1)$ is the evolution operator under the actual Hamiltonian $\widehat{H}(\mathbf{v}, t)$ of the realistic machine. Hence the

difference between these two evaluations are

$$\left|\frac{\partial \mathcal{L}}{\partial \mathbf{v}} - \widehat{\frac{\partial \mathcal{L}}{\partial \mathbf{v}}}\right| = \left|-i\int_0^T \mathrm{d}\tau \, \langle\psi_0| \, \widehat{U}_{\mathbf{v}}^{\dagger}(T,0) M \widehat{U}_{\mathbf{v}}(T,\tau) \left(\frac{\partial H}{\partial \mathbf{v}} - \frac{\partial \widehat{H}}{\partial \mathbf{v}}\right) \widehat{U}(\tau,0) \, |\psi_0\rangle + h.c.\right| \tag{5.2.6.3}$$

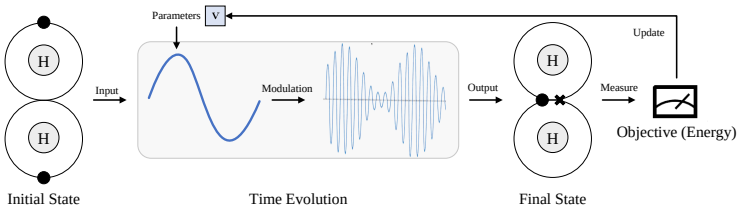
$$\leq 2T\|M\| \max_{\tau\in[0,T]} \left\|\frac{\partial H}{\partial \mathbf{v}}(\mathbf{v},\tau) - \frac{\partial \widehat{H}}{\partial \mathbf{v}}(\mathbf{v},\tau)\right\|, \tag{5.2.6.4}$$

where $\|\cdot\|$ is the spectral norm [134]. □

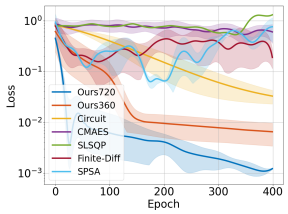
In other words, one can use Algorithm 2 to *optimize the control pulses without a precise understanding of the machine Hamiltonian*, if the difference $H(\mathbf{v},t) - \widehat{H}(\mathbf{v},t)$ varies slowly w.r.t. $\mathbf{v}$ (a rather mild condition to satisfy).

On the contrary, if one relies solely on the Hamiltonian $H(\mathbf{v},t)$ built in the abstract quantum analog machine (e.g., the classical simulation of quantum systems in GRAPE or CRAB), the approximation error could be as large as the difference $H(\mathbf{v},t) - \widehat{H}(\mathbf{v},t)$ itself, a potentially large term compared with its derivative w.r.t. $\mathbf{v}$. This particular advantage of Algorithm 2 exists exactly because of its execution on the real machine, which *is potentially applicable in general scenarios where only approximate machine descriptions are obtainable*.

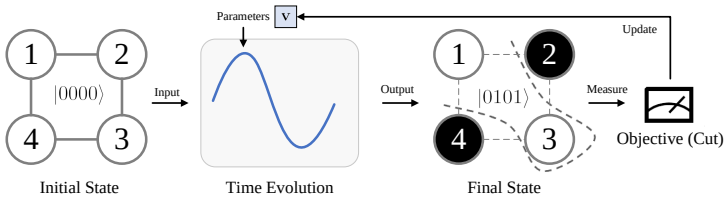
Similar to the circuit case, the systematic error caused by the imprecise evolution under H_j is bounded by the error sum in each evolution under H_j for duration $(1 \pm \frac{3}{4})\pi$, which is usually small.



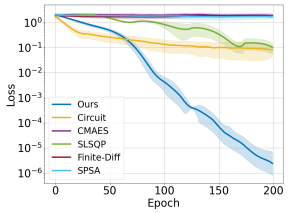
(a) Analog-ansatz-based VQE for the H_2 system



(b) Ground energy search



(c) Analog-ansatz-based QAOA for a MaxCut problem



(d) Cut size maximization

Figure 5.2: Experiments on quantum optimization problems are presented. (a) Experiment results on the H_2 ground energy search. The loss function is the difference between the evaluated energy to the ground energy of H_2, and the lower is the better. Our method with the same (720dt) or less (360dt) pulse duration converges more than 10 times faster than the circuit model [135]. (b) Experiment results on finding the max cut for 4 vertices circular graph. The loss function is the difference between the evaluated cut size to the maximal cut size, the lower the better. Our method outperforms the others by orders of magnitude.

5.3 Applications in quantum optimization

Many important optimization problems in both physics and combinatorics that allow variational solutions can be formulated easily in our framework. For example, finding the ground state of physics systems can be solved by variational quantum eigensolver (VQE) (e.g. [79, 135]), and searching for the max-cut of graphs can be approximated by quantum approximate optimization algorithms (QAOA) [68]. Replacing parameterized circuits by AQAMs in existing variational quantum algorithms leads to significantly improved convergence based on numerical experiments on a classical simulator.

5.3.1 Variational quantum eigensolver

We exhibit our approach via an AQAM comparable to existing circuit VQE in terms of the pulse duration, with significantly better convergence and hardware efficiency in identifying the ground state of the H_2 molecule.

Problem setting. The Hamiltonian of H_2 molecule is expanded with Pauli matrices in the form $H_{\mathrm{H}_2} = \alpha_0 \mathbb{I} + \alpha_1 Z_1 Z_2 + \alpha_2 X_1 X_2 + \alpha_3 Z_1 + \alpha_4 Z_2$, where α_i is a scalar weight calculated in [135]. The ground state $|\psi\rangle$ has the minimal energy, defined by $\mathrm{argmin}_{|\psi\rangle} \langle\psi| H_{\mathrm{H}_2} |\psi\rangle$. This energy function is computed on real machines by Pauli measurements and linearity.

AQAM design. We propose an IBM-AQAM mimicking IBM transmon system [136], and hence realizable on IBM's machines. The IBM-AQAM contains 2 qubits and 4 input pulses $u_{jk}(t)$, and can evolve under

$$H(t) = H_{\text{sys}} + \sum\nolimits_{j,k\in\{0,1\}} \mathcal{M}_{jk}(u_{jk}, t) X_j. \tag{5.3.1.1}$$

Here H_{sys} is a constant Hamiltonian. The input pulses to IBM quantum devices are $u_{jk}(t)$ which are complex values with norm less than 1. These pulses are modulated by the built-in modulation procedure $\mathcal{M}_{jk}$ of the IBM's quantum devices when executed on the real machine. Since the tunable terms have Hamiltonian X_j, we require the IBM-AQAM to be able to evolve under X_j. This is realizable on IBM's machines because in (5.3.1.1), H_{sys} is much weaker than the microwave input pulses for each qubit in form X_j, ensuring a high fidelity simulation of Hamiltonian X_j on real machines. We also require the IBM-AQAM to support initializing in state $|00\rangle$ and measuring with $M = H_{\mathrm{H}_2}$, and these procedures are easy to implement for IBM's machines. We adopt

the parameterization

$$u_{jk}(\mathbf{v}, t) = \mathcal{N}\left(\sum\nolimits_{l=0}^{d}(\mathbf{v}_{jkl0} + i\mathbf{v}_{jkl1}) \cdot P_l\left(\frac{2t}{T} - 1\right)\right) \tag{5.3.1.2}$$

to make the pulse norms less than 1, where $\mathcal{N}(0) = 0, \mathcal{N}(z) = \frac{S(|z|)}{|z|}z$ for $z \neq 0$ is a differentiable normalization function restricting the norm, $S(x) = \frac{1-e^{-x}}{1+e^{-x}}$ is the shifted sigmoid function, T is the duration, and P_l is the l-th Legendre polynomial.

In [135], a parameterized circuit is proposed as layered tunable single qubit rotations and fixed cross-resonance gates, which are compiled to pulses fitting in (5.3.1.1) and sent to the IBM quantum devices. Their one-layer experiments over two qubits have cross resonance gates compiled to pulses with duration around $720\texttt{dt}$ where $\texttt{dt}$=0.22ns. We match it in our experiments, setting $T = 720\texttt{dt}$. Additionally, we test our approach with only half the duration, $T = 360\texttt{dt}$.

Comparisons. We compare our approach to circuit VQE, finite difference method, simultaneous perturbation stochastic approximation (SPSA), and derivative-free methods (CMAES [137] and SLSQP [138]) with the IBM-AQAM on a classical simulator. The experiment results are displayed in Figure 5.2b.

The circuit VQE converges to $\mathcal{L} = 0.02$, while our approach decreases lower: at 400 epoch, with $720\texttt{dt}$ it decreases to less than 0.002, and with $360\texttt{dt}$ it decreses to 0.01. In general, our approach is over 10 times more accurate than the circuit VQE, and 100 times more accurate than the derivative-free methods. The finite difference method and SPSA do not converge because of the intrinsic randomness of quantum measurements at relatively small observation mini-batch ($b_{\text{obs}} = 100$), which would be amplified by the small difference length. With a large enough observation mini-batch, the gradients evaluated by finite difference method has $\sim 3\%$ relative

difference to the gradients evaluated by our approach. These results exhibit the advantages of our differentiable analog framework compared to circuit model and derivative-free analog models.

We note that the benefits of using analog controls in VQE for H_2 molecule are also discovered in a recent result [116], where the optimization is conducted based on GRAPE-like techniques.

5.3.2 Quantum approximate optimization algorithm

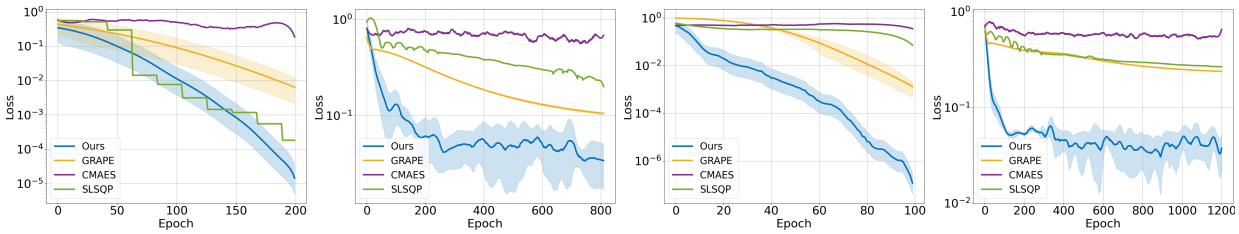
We also investigate the application of QAOA to approximate solutions for the MaxCut problem, an NP-complete problem. With a corresponding AQAM, we achieve a significantly better convergence.

Problem setting. Given a graph $G = \{E, V\}$ where V is the vertex set and $E = \{(i, j)\}$ contains all the edges, our goal is to partition the vertices into two sets (V_0, V_1) so that the number of edges between the two sets is maximized. A cut of an n-node graph G is represented by an n-bit string $s = b_1 b_2 ... b_n$, with $b_j \in \{0, 1\}$ representing in which set the j-th vertex is. We use the computational basis $|s\rangle$ in an n-qubit register to represent the cut s. A maximum cut $|s\rangle$ maximizes the expected cut size $\langle s| C |s\rangle$, where $C = \frac{1}{2} \sum_{(j,k)\in E} C_{j,k}$, $C_{j,k} = \mathbb{I} - Z_j Z_k$. We test the performance on the circular graph shown in the leftmost of Figure 5.2c.

AQAM design. Farhi et al. [68] optimizes a p-layer circuit ansatz $U(\beta, \gamma) = \Pi_{j=1}^{p} e^{-i\beta_j B} e^{-i\gamma_j C}$, where $B = \sum_{j=1}^{n} X_j$. We set $p = 2$ as a baseline.

To have a fair comparison, we design a Cut-AQAM corresponding to the above circuit: $H(t) = \frac{1}{2\pi} \sum_{j=1}^{4} (u_{j0}(t) C_{j,j+1} + u_{j1}(t) X_j)$.

The input pulses are real functions $u_{jk}(t)$, where we restrict the energy input by requiring



(a) $|+\rangle$ preparation (b) $|\Phi^+\rangle$ preparation (c) X gate synthesis (d) CNOT gate synthesis

Figure 5.3: Experimental results on quantum control are presented. We apply our differentiable analog quantum computing framework to state preparation and gate synthesis. Four methods (SLSQP, CMAES, GRAPE, and ours) are used in these tasks. In gate synthesis, the accuracy of the built-in gates from IBM Qiskit is shown in dashed lines. Our method outperforms the other three algorithms in terms of convergence rate and accuracy by up to orders of magnitude.

$|u_{jk}| \leq 1$. Cut-AQAM natively supports evolving merely under $C_{j,j+1}$ or X_j by setting u_{jk} as indicator functions. We also require it to support initializing in state $|0\rangle$ and measuring with observable C. In our experiment, we set the duration $T = 4$ within which the circuit QAOA can be realized by Cut-AQAM. Similar to (5.3.1.2), we parameterize $u_{jk}(\mathbf{v}, t)$ by a normalized linear combination of Legendre's polynomial.

Comparisons. The experiments are set up in a similar way as in Section 5.3.1. Results are shown in Figure 5.2d. The circuit QAOA and SLSQP converge to 0.08 at 200 epoch, while our method converges to 2.6×10^{-6}. The finite difference method, SPSA, and CMAES do not converge to a value less than 1. The analysis is similar to Section 5.3.1. Since Cut-AQAM does not have a high-frequency modulation, SLSQP also converges close to 0 but is slower than our approach.

5.4 Applications in quantum control

Quantum control problems fall into two categories: 1) *state preparation*: to steer a given initial state into a target final state; 2) *gate synthesis*: to effect a specific unitary transformation (quantum gate) in the system. In what follows, we discuss how to formulate and solve quantum

control problems using our differentiable analog quantum computing framework. We demonstrate our methodology by numerical experiments on the IBM-AQAM.

5.4.1 State preparation

To prepare the target state $|\psi_{tar}\rangle$ from certain fixed initial state, we desire a parameter vector $\mathbf{v}$ that minimizes the loss function defined in (5.2.2.1) with the observable $M = \mathbb{I} - |\psi_{tar}\rangle \langle\psi_{tar}|$. We consider two tasks: 1) to prepare the state $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ from $|0\rangle$; 2) to prepare the two-qubit maximally entangled state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ from $|00\rangle$. $|\Phi^+\rangle =$. In both tasks, the loss function is readily computed as the measurement merely involves local Pauli operators, and can be carried out on the IBM-AQAM.

In the numerical experiments, the pulse duration is fixed as $T = 20\mathtt{dt}$ for the $|+\rangle$ state, and $T = 1200\mathtt{dt}$ for the $|\Phi^+\rangle$ state. We also compare our method with two gradient-free methods (SLSQP, CMAES) and the GRAPE algorithm. In both tasks, ours achieves faster convergence than all other three methods. In the $|+\rangle$ task, the final loss value from our method reads approximately 10^{-5}, which is 18 times better than the second best result (i.e., SLSQP), as in in Figure 5.3 (a). In the $|\Phi^+\rangle$ task, the final loss value from our method reads 0.034, which is 3 times better than GRAPE and 6 times better than SLSQP, as shown in Figure 5.3 (b).

5.4.2 Gate synthesis

Gate synthesis is more challenging than state preparation because we hope to engineer a full unitary gate U_{tar} instead of just mapping a single state to a target state. To this end we first specify a set of pairs $\mathcal{S} = \{(|\mathbf{x}_j\rangle, |\mathbf{y}_j\rangle)\}_{j=1}^{k}$ that completely determines U_{tar} in the sense that

no unitary map U other than U_{tar} satisfies $\left| \langle \mathbf{y}_j | U | \mathbf{x}_j \rangle \right| = 1$ for all $j = 1, 2, .., k$. Then, we consider the loss function $\mathcal{L} = \frac{1}{k} \sum_{j=1}^{k} \mathcal{L}_j$, where each $\mathcal{L}_j$ is defined as in (5.2.2.1) with quantum observable $M = \mathbb{I} - |\mathbf{y}_j\rangle \langle \mathbf{y}_j|$ and initial state $|\mathbf{x}_j\rangle$. When $\mathcal{L}(\mathbf{v})$ is close to 0, the time evolution controlled by the parameter $\mathbf{v}$ is approximately U_{tar}.

The X gate and CNOT gate are widely used in digital quantum computing and supported by IBM Qiskit. We now exemplify our method by recovering these two gates on the IBM-AQAM. In both cases, the loss function can be readily computed on IBM machines.

The pulse duration is chosen to be comparable to the built-in ones in IBM Qiskit: $T = 160\texttt{dt}$ for X gate, and $T = 1200\texttt{dt}$ for CNOT gate. In IBM Qiskit, the X gate is implemented by a pulse of duration $T = 160\texttt{dt}$, and CNOT is by $T = 1056\texttt{dt}$. We apply four methods (SLSQP, CMAES, GRAPE, ours) to the gate synthesis tasks. The results are shown in Figure 5.3 (c), (d). In the synthesis of X gate, the final loss value from our method is 1.17×10^{-7}, which is over 10^4 times more accurate than the other three methods. In the more involved task of synthesizing CNOT gate, our method returns a pulse sequence with loss 0.0172, while the results from other methods are no less than 0.2. It is worth noting that the loss of IBM built-in calibrated pulses evaluated on IBM machines are 0.019 for X gate and 0.043 for CNOT gate when no measurement error mitigation or state preparation error mitigation technique is applied.

Part III

Verify COQC Software Stacks

Chapter 6: Part III Preface

In the process of building a software stack, human errors are inevitable. Moreover, bugs in quantum programs are detrimental. Because every quantum algorithm makes use of the superposition of quantum states, slight human errors in the program almost always generate non-negligible effects on the output, and most of the time the output measurement distribution is greatly different than the ideal distribution. However, locating the bug in an erroneous program is utterly difficult. Traditional techniques like software testing no longer suffice for debugging quantum programs (more details in Section 7.1), which propels us to explore alternative ways to ensure the correctness of quantum programs in the software stack.

We study formal verification for quantum programs in this part, which has several unique advantages in ensuring quantum program correctness. Techniques in formal verification normally require special expertise and many human efforts. Meanwhile, the established correctness of programs from formal verification is relatively hard to be transferred to other programs. However, they provide strong guarantees of the correctness of programs. To validate the feasibility of applying formal verification to quantum programs, we target mature frameworks of quantum programs. Since the foundational software for HOQC is relatively new (details in Part II), we shift our focus to software stacks for COQC in this part.

Similar to the HOQC software stacks like Chapter 3, COQC software stacks contain several

major components: (1) high-level programs implementing quantum algorithms; (2) low-level programs generating quantum circuits; (2) COQC compilers to map the circuits to pulses. Many prototypes are developed in the literature [16, 17]. We develop formal methods for the abstract version of these tools to ensure their correctness.

In Chapter 7, we investigate deductive program verification for implementations of quantum algorithms. We target Shor's factorization algorithm, the most complicated quantum algorithm, to justify the feasibility of deductive verification for ensuring the semantics correctness of quantum programs for high-level and low-level descriptions of quantum algorithms. This chapter is based on [139]: Peng, Y., Hietala, K., Tao, R., Li, L., Rand, R., Hicks, M., & Wu, X. (2023). A formally certified end-to-end implementation of Shor's factorization algorithm. *Proceedings of the National Academy of Sciences, 120*(21), e2218775120.

In Chapter 8, we study an algebraic reasoning method for quantum while programs. It leverages non-idempotent Kleene algebra to formally verify program rewrite rules. With verified rewrite rules, compilers can ensure the correctness of their program transformations and reduce resource consumption in compiled programs. This chapter is based on [140]: Peng, Y., Ying, M., & Wu, X. (2022, June). Algebraic reasoning of Quantum programs via non-idempotent Kleene algebra. *In Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (pp. 657-670).

Chapter 7: Deductive Verification for Shor's Factorization Algorithm

Chapter summary

While hardware errors have garnered significant attention as the major obstacle to quantum computing, the error due to human factors in the implementation is less recognized. We identify human programming errors as another important source of errors, whereas most existing techniques from the classical domain fail to transfer at scale to quantum programming. The adaptation of formal methods to quantum programming is proposed as a solution that circumvents quantum unique challenges and leads to the high-assurance implementation of large-scale quantum applications in a principle way.

Deductive program verification, where programmers not only write programs but also mechanized specification of their programs and proofs, suits well for verifying the correctness of quantum programs. As a demonstration of the feasibility, in this chapter, we present a formally certified end-to-end implementation of Shor's prime factorization algorithm due to its complexity and importance.

7.1 Introduction

As developments in quantum computer hardware bring promising quantum applications closer to reality, a key question to contend with is: *How can we be sure that a quantum computer program, when executed, will give the right answer?* A well-recognized threat to correctness is the quantum computer hardware, which is susceptible to decoherence errors. Techniques to provide hardware-level fault tolerance are under active research [141, 142]. A less recognized threat comes from errors—*bugs*—in the program itself, as well as errors in the software that prepares a program to run on a quantum computer (compilers, linkers, etc.). In the classical domain, program bugs are commonplace and are sometimes the source of expensive and catastrophic failures or security vulnerabilities.

Quantum programs that provide a performance advantage over their classical counterparts are even more challenging to write, understand and certify (Figure 7.1 (a)). They often involve the use of randomized algorithms and leverage unfamiliar quantum-specific concepts, including superposition, entanglement, and destructive measurement. Quantum programs are also hard to test. To debug a failing test, programmers cannot easily observe (measure) an intermediate state due to the destructive nature of quantum measurement. Moreover, many quantum algorithms generate samples over an exponentially large output domain, whose statistical properties could require exponentially many samples to be verified information-theoretically. Simulating a quantum program on a classical computer can help, but is limited by such computers' ability to faithfully represent a quantum state of even modest size (which is why we must build quantum hardware). The fact that near-term quantum computers are error-prone adds another layer of difficulty.

Proving programs correct with formal methods

As a potential remedy to these problems, we have been exploring how to use *formal methods* (aka *formal verification*) to develop quantum programs (Figure 7.1 (b)). Formal methods are processes and techniques by which one can mathematically *prove* that software does what it should, for all inputs; the proved-correct artifact is referred to as *formally certified*. The formal verification is usually conducted by using a *proof assistant*, which is a software tool for formalizing mathematical definitions and stating and proving properties about them. A proof assistant may produce proofs automatically or assist a human in doing so, interactively. Either way, the proof assistant confirms that a proof is correct by employing a proof verifier. Most modern proof assistants implement proof verification by leveraging the *Curry-Howard correspondence*, which embodies a surprising and powerful analogy between formal logic and programming language type systems [143, 144]. Automating and confirming such proofs have, for more than 50 years, been a grand challenge for computing research [145].

While early developments of formal methods led to disappointment [146], the last two decades have seen remarkable progress. Notable successes include the development of the seL4 microkernel [147] and the CompCert C compiler [148]. For the latter, the benefits of formal methods have been demonstrated empirically: Using sophisticated testing techniques, researchers found hundreds of bugs in the popular mainstream C compilers `gcc` and `clang`, but none in CompCert's verified core [149]. Formal methods have also been successfully deployed to prove major mathematical theorems (e.g., the Four Color theorem [150]) and build computer-assisted proofs in the grand unification theory of mathematics [151, 152].

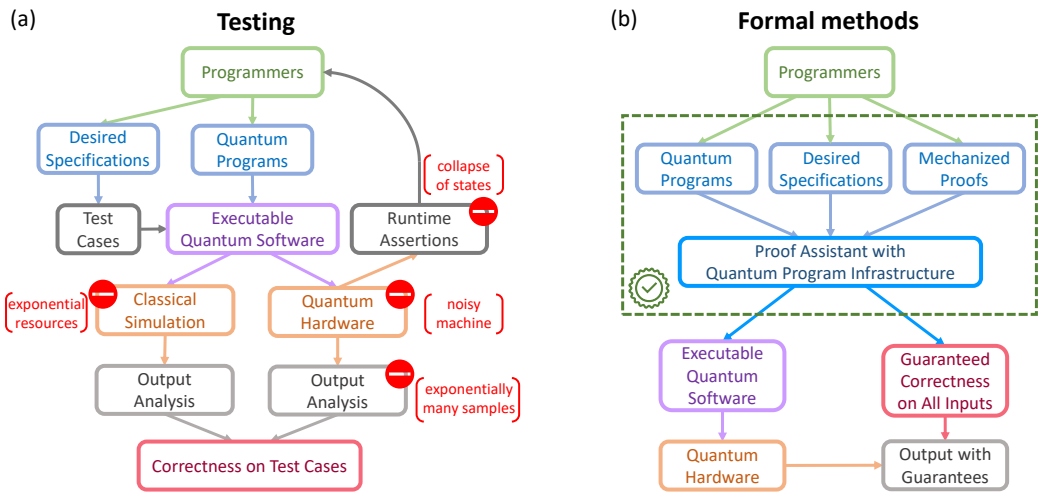


Figure 7.1: Comparison of developing quantum programs with (a) testing and with (b) deductive program verification. The major classical routines of debugging with testing are blocked by quantum-specific features, and can only guarantee correctness in test cases. In the formal methods approach, a proof assistant will mechanically certify quantum programs along with their specifications and proofs. If successful, the certified implementation is guaranteed to meet the desired specifications on all possible inputs, even without running the program on real machines.

Formal methods for quantum programs

Our key observation is that the symbolic reasoning that underlies formal methods is not limited by the aforementioned difficulties of testing directly on quantum machines or classically simulating them. As a result, it may be a viable alternative to certifying the correctness of quantum programs. Our research has explored how to put this observation into practice. Precisely, using the Coq proof assistant [153], we defined a *simple quantum intermediate representation* [3] (SQIR) for expressing a quantum program as a series of operations—essentially a kind of circuit—and specified those operations' mathematical meaning. Thus we can state the mathematical properties of an SQIR program and prove that they always hold without needing to run that program. Assured that the program is correct, we can run it on specific inputs by asking Coq to *extract* the SQIR program to a OpenQASM 2.0 circuit, and then run it on a real machine.

Adapting formal methods developed for classical programs to work on quantum ones is conceptually straightforward but pragmatically challenging. Consider that classical program states are (in the simplest terms) maps from addresses to bits (0 or 1); thus, a state is essentially a length-n vector of Booleans. Quantum states are much more involved: In SQIR an n-qubit quantum state is represented as a length-2^n vector of complex numbers and the meaning of an n-qubit operation is represented as a $2^n \times 2^n$ matrix—applying an operation to a state is tantamount to multiplying the operation's matrix with the state's vector. Proofs over all possible inputs thus involve translating such multiplications into symbolic formulae and then reasoning about them.

Given the potentially large size of quantum states, such formulae could become quite large and difficult to reason about. To cope, we developed automated *tactics* to translate symbolic states into normalized algebraic forms, making them more amenable to automated simplification. We also eschew matrix-based representations entirely when an operation can be expressed symbolically in terms of its action on basis states. With these techniques and others [154], we proved the correctness of key components of several quantum algorithms—Grover's search algorithm [155] and quantum phase estimation (QPE) [2]—and demonstrated advantages over competing approaches [5, 156, 157, 158].

With this promising foundation in place, several challenges remain. First, both Grover's and QPE are parameterized by *oracles*, which are classical algorithmic components that must be implemented to run on quantum hardware. These must be reasoned about, too, but they can be large (many times larger than an algorithm's quantum scaffold) and can be challenging to encode for quantum processing, bug-free. Another challenge is proving the end-to-end properties of hybrid quantum/classical algorithms. These algorithms execute code on both classical and quantum computers to produce a final result. Such algorithms are likely to be common in near-

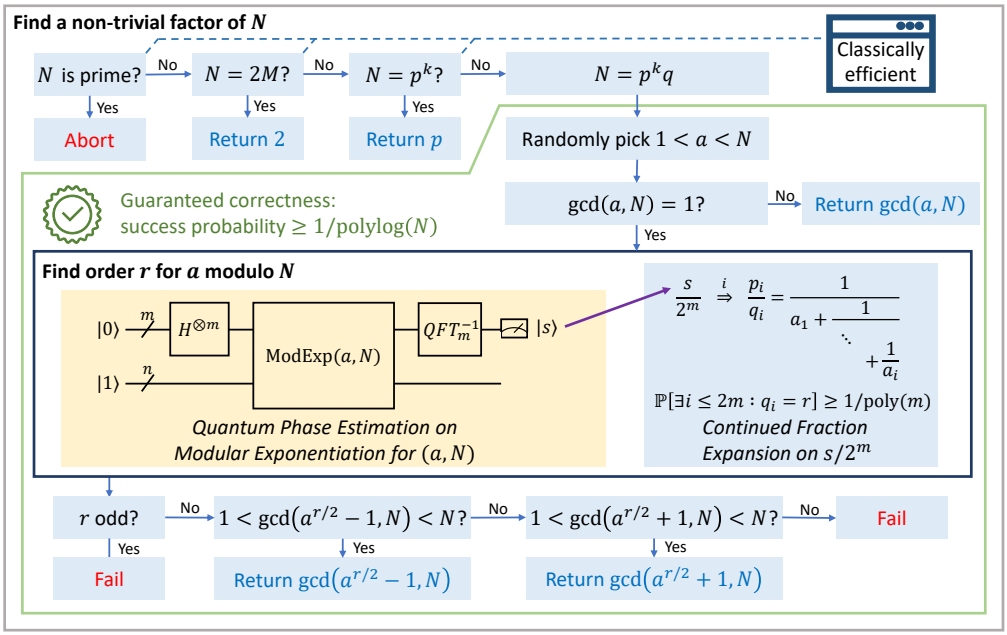


Figure 7.2: Overview of Shor's factoring algorithm, which finds a non-trivial factor of integer N (not 1 or N) with high probability in polynomial time. This is a quantum-classical hybrid algorithm, whose quantum part (marked cream) is a sub-procedure of finding multiplicative orders (enclosed in the blue frame). We implement and mechanically certified Shor's algorithm (enclosed in the green frame), for N not prime, even, or a prime power (these cases can be efficiently tested for and solved by classical algorithms). A detailed illustration of our implementation of $\text{ModExp}(a, N)$ is in Figure 7.4.

term deployments in which quantum processors complement classical ones. Finally, end-to-end certified software must implement and reason about probabilistic algorithms, which are correct with a certain probability and may require multiple runs.

To close these gaps, and thereby demonstrate the feasibility of the application of formal methods to quantum programming, we have produced a fully certified version of Shor's prime factorization algorithm [2], which is famous for breaking widely-used RSA cryptographic systems. This algorithm has been a fundamental motivation for the development of quantum computers and is at a scale and complexity not reached in prior formalization efforts.

As shown in Figure 7.2, Shor developed a sophisticated, quantum-classical hybrid algorithm to factor a number N: the key quantum part—*order finding*—preceded and followed by

classical computation—primality testing before, and conversion of found orders to prime factors, after. The algorithm's correctness proof critically relies on arguments about both its quantum and classical parts, and also on several number-theoretical arguments.

While it is difficult to factor a number, it is easy to confirm a proposed factorization (the factoring problem is inside the NP complexity class). One might wonder: why prove a program correct if we can always efficiently check its output? When the check shows an output is wrong, this fact does not help with computing the correct output and provides no hint about the source of the implementation error. By contrast, formal verification allows us to identify the source of the error: it is precisely in the subprogram that we could not certify.

Moreover, because inputs are reasoned about *symbolically*, the complexity of all-input certification can be (much) less than the complexity of single-output correctness checking. For example, one can symbolically verify that a quantum circuit generates a uniform distribution over n bits, but directly checking whether the output samples from a uniform distribution over n bits could take as many as $2^{\Theta(n)}$ samples [159]. As such, with formal methods, one could potentially certify implementations for major quantum applications, like quantum simulation which is BQP-complete [57] and believed to lie outside NP.

An instantiation of the scheme in Figure 7.1 (b) for Shor's algorithm is given in Figure 7.3 (a)(b). We have certified the implementation against both a correctness specification (e.g., the likelihood of success) and a resource specification (e.g., the gate count) (see Figure 7.6).

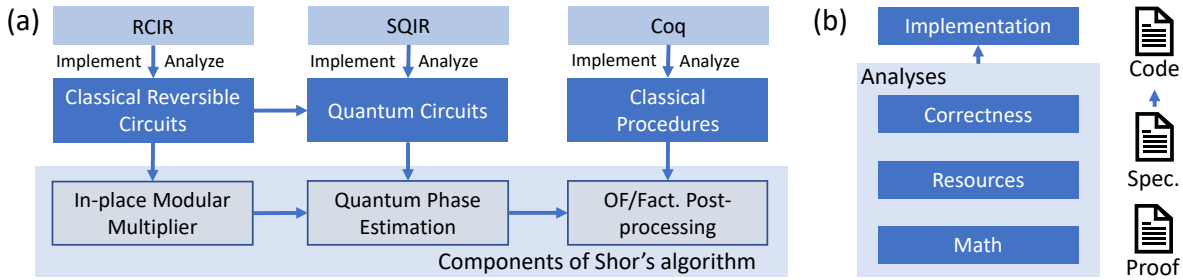


Figure 7.3: Technical illustration of our fully certified implementation of Shor's algorithm. (a) The schematic framework of our implementation in Coq. Intermediate representations SQIR and RCIR are embedded in Coq, dealing with classical reversible and quantum circuits respectively. (b) An instantiation of the formal methods scheme in Shor's implementation.

7.2 End-to-end implementation of Shor's algorithm

Shor's factorization algorithm consists of two parts. The first employs a hybrid classical-quantum algorithm to solve the order-finding problem; the second reduces factorization to order-finding. In this section, we present an overview of Shor's algorithm (see Figure 7.2 for a summary). In the next sections, we discuss details about our implementation (see Figure 7.3) and certified correctness properties.

We give a brief introduction to the procedure of Shor's algorithm, whose details are presented later. The classical pre-processing will identify cases where N is prime, even, or a prime power, which can be efficiently tested for and solved by classical algorithms. Otherwise, one will proceed to the main part of Shor's algorithm (enclosed in the green frame) to solve the case where $N = p^k q$. One starts with a random integer sample a between 1 and N. When a is a co-prime of N, i.e., the greatest common divisor $\gcd(a, N) = 1$, the algorithm leverages a quantum computer and classical post-processing to find the order r of a modulo N (i.e., the smallest positive integer r such that $a^r \equiv 1(\mod N)$). The quantum part of order finding involves quantum phase estimation (QPE) on modular multipliers for (a, N). The classical post-processing finds the con-

tinued fraction expansion (CFE) $[a_1, a_2, \cdots, a_{2m}]$ of the output $s/2^m \approx k/r$ of quantum phase estimation to recover the order r. Further classical post-processing will rule out cases where r is odd before outputting the non-trivial factor. To formally prove the correctness of the implementation, we first prove separately the correctness of the quantum component (i.e., QPE with in-place modular multiplier circuits for any (a, N) on n bits) and the classical component (i.e., the convergence and the correctness of the CFE procedure). We then integrate them to prove that with one randomly sampled a, the main part of Shor's algorithm, i.e., the quantum order-finding step sandwiched between the pre and post classical processing, will succeed in identifying a non-trivial factor of N with probability at least 1/polylog(N). By repeating this procedure polylog(N) times, our certified implementation of Shor's algorithm is guaranteed to find a non-trivial factor with a success probability close to 1.

Our focus is on implementation over an ideal quantum computer, where there is no gate implementation error and no topology constraints. In practice, error correction codes become necessary to correct errors caused by system decoherence or state leakage. This involves using multiple physical qubits to represent a logical qubit, which can significantly increase the circuit size depending on the precision of the quantum computer. Incorporating error correction into a certified implementation of Shor's algorithm is left as a future research direction.

7.2.1 A hybrid algorithm for order finding

The multiplicative order of a modulo N, represented by ord(a, N), is the least integer r larger than 1 such that $a^r \equiv 1 \pmod{N}$. Calculating ord(a, N) is hard for classical computers, but can be efficiently solved with a quantum computer, for which Shor proposed a hybrid

classical-quantum algorithm [2]. This algorithm has three major components: (1) in-place modular multiplication on a quantum computer; (2) quantum phase estimation; and (3) continued fraction expansion on a classical computer.

In-place Modular Multiplication An in-place modular multiplication operator $IMM(a, N)$ on n working qubits and s ancillary qubits satisfies the following property:

$$\forall x < N, \;\; IMM(a, N)|x\rangle_n|0\rangle_s = |(a \cdot x) \text{ mod } N\rangle_n|0\rangle_s,$$

where $0 < N < 2^{n-1}$. It is required that a and N are co-prime, otherwise the operator is irreversible. This requirement implies the existence of a multiplicative inverse a^{-1} modulo N such that $a \cdot a^{-1} \equiv 1 \pmod{N}$.

Quantum Phase Estimation Given a subroutine U and an eigenvector $|\psi\rangle$ with eigenvalue $e^{i\theta}$, quantum phase estimation (QPE) finds the closest integer to $\frac{\theta}{2\pi}2^m$ with high success probability, where m is a predefined precision parameter.

Shor's algorithm picks a random a from $[1, N)$ first, and applies QPE on $IMM(a, N)$ on input state $|0\rangle_m |1\rangle_n |0\rangle_s$ where $m = \lfloor \log_2 2N^2 \rfloor, n = \lfloor \log_2 2N \rfloor$ and s is the number of ancillary qubits used in $IMM(a, N)$. Then a computational basis measurement is applied on the first m qubits, generating an output integer $0 \leq \texttt{out} < 2^m$. The distribution of the output has ord(a, N) peaks, and these peaks are almost equally spaced. We can extract the order by the following procedure.

Continued Fraction Expansion The post-processing of Shor's algorithm invokes the continued

fraction expansion (CFE) algorithm. A k-level continued fraction is defined recursively by

$$\langle\rangle = 0,$$
$$\langle a_1, a_2, ..., a_k\rangle = \frac{1}{a_1 + \langle a_2, a_3, ..., a_k\rangle}.$$

k-step CFE finds a k-level continued fraction to approximate a given real number. For a rational number $0 \leq \frac{a}{b} < 1$, the first term of the expansion is $\lfloor \frac{b}{a} \rfloor$ if $a \neq 0$, and we recursively expand $\frac{b \bmod a}{a}$ for at most k times to get an approximation of $\frac{a}{b}$ by a k-level continued fraction. In Coq, the CFE algorithm is implemented as

```
Fixpoint CFE_ite (k a b p1 q1 p2 q2 : ℕ) : ℕ × ℕ :=
  match k with
  | 0 ⇒ (p1, q1)
  | S k' ⇒ if a = 0 then (p1, q1)
           else let (c, d) := (⌊b/a⌋, b mod a) in
           CF_ite k' d a (c·p1 + p2) (c·q1 + q2) p1 q1
  end.
Definition CFE k a b := snd (CF_ite (k+1) a b 0 1 1 0).
```

Function `CFE_ite` takes in the number of iterations k, target fraction `a/b`, the fraction from the $(k-1)$-step expansion, and the $(k-2)$-step expansion. Function `CFE k a b` represents the denominator in the simplified fraction equal to the k-level continued fraction that is the closest to $\frac{a}{b}$.

The post-processing of Shor's algorithm expands $\frac{\texttt{out}}{2^{\texttt{m}}}$ using CFE, where `out` is the measurement result and `m` is the precision for QPE defined above. It finds the minimal step `k` such that $a^{\texttt{CFE k out } 2^{\texttt{m}}} \equiv 1 \pmod{N}$ and $k \leq 2m+1$. With probability no less than $1/\text{polylog}(N)$, there exists k such that `CFE k out` $2^{\texttt{m}}$ is the multiplicative order of a modulo N. We can repeat

the QPE and post-processing for polylog(N) times. Then the probability that the order exists in one of the results can be arbitrarily close to 1. The minimal valid post-processing result is highly likely to be the order.

7.2.2 Reduction from factorization to order finding

To completely factorize composite number N, we only need to find one non-trivial factor of N (i.e., a factor that is not 1 nor N). If a non-trivial factor d of N can be found, we can recursively solve the problem by factorizing d and $\frac{N}{d}$ separately. Because there are at most $\log_2(N)$ prime factors of N, this procedure repeats for at most polylog(N) times. A classical computer can efficiently find a non-trivial factor in the case where N is even or $N = p^k$ for prime p. However, Shor's algorithm is the only known (classical or quantum) algorithm to efficiently factor numbers for which neither of these is true.

Shor's algorithm randomly picks an integer $1 \leq a < N$. If the greatest common divisor gcd(a, N) of a and N is a non-trivial factor of N, then we are done. Otherwise we invoke the hybrid order finding procedure to find ord(a, N). With probability no less than one half, one of gcd $\left(a^{\lfloor \frac{\text{ord}(a,N)}{2} \rfloor} \pm 1, N\right)$ is a non-trivial factor of N. Note that gcd $\left(a^{\lfloor \frac{\text{ord}(a,N)}{2} \rfloor} \pm 1, N\right)$ can be efficiently computed by a classical computer[160]. By repeating the random selection of a and the above procedure for constant times, the success probability to find a non-trivial factor of N is close to 1.

7.2.3 Implementation details

7.2.3.1 Implementation of in-place modular exponentiation

One of the pivoting components of Shor's order-finding procedure is a quantum circuit for in-place modular exponentiation, effectively in-place modular multiplication (IMM). We initially tried to define this operation in SQIR but found that for purely classical operations (that take basis states to basis states), SQIR's general quantum semantics make proofs unnecessarily complicated. In response, we developed the *reversible circuit intermediate representation* (RCIR) to express classical functions and prove their correctness. RCIR programs can be translated into SQIR, and we prove this translation correct.

RCIR

RCIR contains a universal set of constructs on classical bits labeled by natural numbers. The syntax is:

$$R := \texttt{skip} \mid \texttt{X}\, n \mid \texttt{ctrl}\, n\, R \mid \texttt{swap}\, m\, n \mid R_1;\ R_2.$$

Here `skip` is a unit operation with no effect, `X` n flips the n-th bit, `ctrl` n R executes subprogram R if the n-th bit is 1 and otherwise has no effect, `swap` m n swaps the m-th and n-th bits, and $R_1; R_2$ executes subprograms R_1 and R_2 sequentially. We remark that `swap` is not necessary for the expressiveness of the language, since it can be decomposed into a sequence of three `ctrl` and `X` operations. We include it here to facilitate `swap`-specific optimizations of the circuit.

As an example, we show the RCIR code for the MAJ (majority) operation [161], which is an essential component of the ripple-carry adder.

```
Definition MAJ a b c :=
  ctrl c (X b) ; ctrl c (X a) ; ctrl a (ctrl b (X c)).
```

It takes in three bits labeled by a, b, c whose initial values are v_a, v_b, v_c correspondingly, and stores v_a `xor` v_c in a, v_b `xor` v_c in b, and $MAJ(v_a, v_b, v_c)$ in c. Here $MAJ(v_a, v_b, v_c)$ is the majority of v_a, v_b and v_c, the value that appears at least twice.

To reverse a program written in this syntax, we define a reverse operator by $\texttt{skip}^{\texttt{rev}} = \texttt{skip}$, $(\texttt{X}\ n)^{\texttt{rev}} = \texttt{X}\ n$, $(\texttt{ctrl}\ n\ R)^{\texttt{rev}} = \texttt{ctrl}\ n\ R^{\texttt{rev}}$, $(\texttt{swap}\ m\ n)^{\texttt{rev}} = \texttt{swap}\ m\ n$, $(R_1;\ R_2)^{\texttt{rev}} = R_2^{\texttt{rev}};\ R_1^{\texttt{rev}}$. We prove that the reversed circuit will cancel the behavior of the original circuit.

We can express the semantics of a RCIR program as a function between Boolean registers. We use notation $[k]_n$ to represent an n-bit register storing natural number $k < 2^n$ in binary representation. Consecutive registers are labeled sequentially by natural numbers. If $n = 1$, we simplify the notation to $[0]$ or $[1]$.

The translation from RCIR to SQIR is natural since every RCIR construct has a direct correspondence in SQIR. The correctness of this translation states that the behavior of a well-typed classical circuit in RCIR is preserved by the generated quantum circuit in the context of SQIR. That is, the translated quantum circuit turns a state on the computational basis into another one corresponding to the classical state after the execution of the classical reversible circuit.

Details of IMM

Then the goal is to construct a reversible circuit $IMM_c(a, N)$ in RCIR satisfying

$$\forall x < N,\ \ [x]_n[0]_s \xrightarrow{IMM_c(a,N)} [a \cdot x \bmod N]_n[0]_s. \tag{7.2.3.1}$$

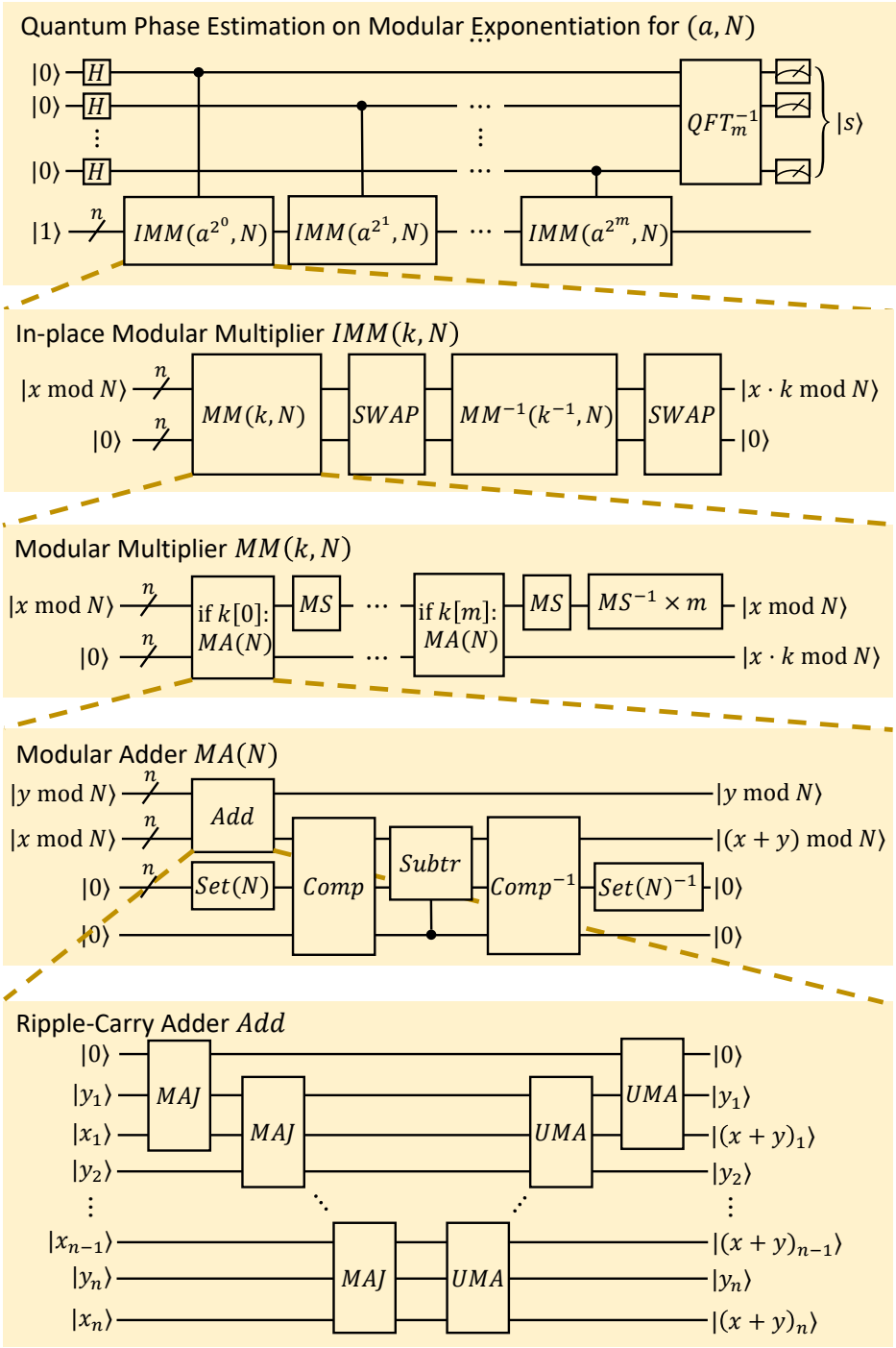


Figure 7.4: An overview of our certified implementation of modular exponentiation circuit.

so that we can translate it into a quantum circuit in SQIR. The overview of our implementation of such a circuit is presented in Figure 7.4. We present it in detail below.

Since we will encounter bit-level logic frequently, we define $x \odot_? y$ for binary logic operator $\odot$ as a binary expression over x and y whose value is 1 if $x \odot y$ is true and 0 otherwise. For example, $3 \geq_? 2 = 1$ and $3 \leq_? 2 = 0$ since $3 \geq 2$. This notation also carries over into our Coq code, e.g. `x<?y` is the expression $x <_? y$.

Adapting the standard practice [162], we implement modular multiplication based on repeated modular additions. For addition, we use Cuccaro et al.'s ripple-carry adder (RCA) [161]. RCA realizes the transformation

$$[c][x]_n[y]_n \xrightarrow{RCA} [c][x]_n[(x + y + c) \bmod 2^n]_n,$$

for ancillary bit $c \in \{0, 1\}$ and inputs $x, y < 2^{n-1}$. We use Cucarro et al.'s RCA-based definitions of subtractor (SUB) and comparator (CMP), and we additionally provide a n-qubit register swapper (SWP) and shifter (SFT) built using swap gates. These components realize the following transformations:

$$[0][x]_n[y]_n \xrightarrow{SUB} [0][x]_n[(y - x) \bmod 2^n]_n$$

$$[0][x]_n[y]_n \xrightarrow{CMP} [x \geq_? y][x]_n[y]_n$$

$$[x]_n[y]_n \xrightarrow{SWP} [y]_n[x]_n$$

$$[x]_n \xrightarrow{SFT} [2x]_n$$

We remark here that SFT is correct only when $x < 2^{n-1}$. With these components, we can build

a modular adder (`ModAdd`) and modular shifter (`ModSft`) using two ancillary bits at positions 0 and 1.

```
Definition ModAdd n :=
    SWP02 n; RCA n; SWP02 n; CMP n;
    ctrl 1 (SUB n); SWP02 n; (CMP n)^rev; SWP02 n.
Definition ModSft n := SFT n; CMP n; ctrl 1 (SUB n).
```

$\texttt{SWP}_{02}$ is the register swapper applied to the first and third n-bit registers. These functions realize the following transformations:

$$[0][0][N]_n[x]_n[y]_n \xrightarrow{\texttt{ModAdd}} [0][0][N]_n[x]_n[(x+y) \text{ mod } N]_n$$

$$[0][0][N]_n[x]_n \xrightarrow{\texttt{ModSft}} [0][N \leq_? 2x][N]_n[2x \text{ mod } N]_n$$

Note that $(a \cdot x)$ mod N can be decomposed into

$$(a \cdot x) \text{ mod } N = \left(\sum_{i=0}^{n-1} (1 \leq_? a_i) \cdot 2^i \cdot x \right) \text{ mod } N,$$

where a_i is the i-th bit in the little-endian binary representation of a. By repeating `ModSft`s and `ModAdd`s, we can perform $(a \cdot x)$ mod N according to this decomposition, eventually generating a circuit for modular multiplication on two registers ($MM(a, N)$), which implements

$$[x]_n[0]_n[0]_s \xrightarrow{MM(a,N)} [x]_n[a \cdot x \text{ mod } N]_n[0]_s.$$

Here s is the number of additional ancillary qubits, which is linear to n. Finally, to make the operation in-place, we exploit the modular inverse a^{-1} modulo N:

```
Definition IMM a N n :=
```

```
MM a N n; SWP01 n; (MM a^-1 N n)^rev.
```

There is much space left for optimization in this implementation. Other approaches in the literature [163, 164, 165, 166, 167] may have a lower depth or fewer ancillary qubits. We chose this approach because its structure is cleaner to express in our language, and its asymptotic complexity is feasible for efficient factorization, which makes it great for mechanized proofs.

7.2.3.2 Implementation of Shor's algorithm

Our final definition of Shor's algorithm in Coq uses the `IMM` operation along with a `SQIR` implementation of `QPE` described in the previous sections. The quantum circuit to find the multiplicative order $\mathrm{ord}(a, N)$ is then

```
Definition shor_circuit a N :=
  let m := log2 (2*N^2) in
  let n := log2 (2*N) in
  let f i := IMM (modexp a (2^i) N) N n in
  X (m + n - 1); QPE m f.
```

We can extract the distribution of the result of the random procedure of Shor's factorization algorithm

```
Definition factor (a N r : ℕ) :=
  let cand1 := Nat.gcd (a ^ (r / 2) - 1) N in
  let cand2 := Nat.gcd (a ^ (r / 2) + 1) N in
  if (1 <? cand1) && (cand1 <? N) then Some cand1
  else if (1 <? cand2) && (cand2 <? N) then Some cand2
  else None.
Definition shor_body N rnd :=
  let m := log2 (2*N^2) in
  let k := 4*log2 (2*N)+11 in
  let distr := join (uniform 1 N)
```

```
                (fun a ⇒ run (to_base_ucom (m+k)
                          (shor_circuit a N))) in
  let out := sample distr rnd in
  let a := out / 2^(m+k)) in
  let x := (out mod (2^(m+k))) / 2^k in
  if Nat.gcd a N =? 1%ℕ
  then factor a N (OF_post a N x n)
  else Some (Nat.gcd a N).
Definition end_to_end_shor N rnds :=
  iterate rnds (shor_body N).
```

Here `factor` is the reduction finding non-trivial factors from multiplicative order, `shor_body` generates the distribution and sampling from it, and `end_to_end_shor` iterates `shor_body` for multiple times and returns a non-trivial factor if any of them succeeds.

7.3 Certification of the implementation

With the properties and correctness of order finding established, we can prove the success probability of the algorithm overall. In this section, we summarize the facts we have proved in Coq to fully verify Shor's algorithm.

For the overall algorithm, we prove that the order finding procedure combined with the classical post-processing will output a non-trivial factor with a success probability of at least $2e^{-2}/\pi^2\lfloor\log_2(N)\rfloor^4$, which is exactly half of the success probability of order finding. Namely, we prove that for at least a half of the integers a between 1 and N, the order r will be even and either $\gcd(a^{r/2}+1, N)$ or $\gcd(a^{r/2}-1, N)$ will be a non-trivial factor of N. Shor's original proof [2] of this property made use of the existence of the group generator of $\mathbb{Z}_{p^k}$, also known as primitive roots, for odd prime p. However, the known proof of the existence of primitive roots is

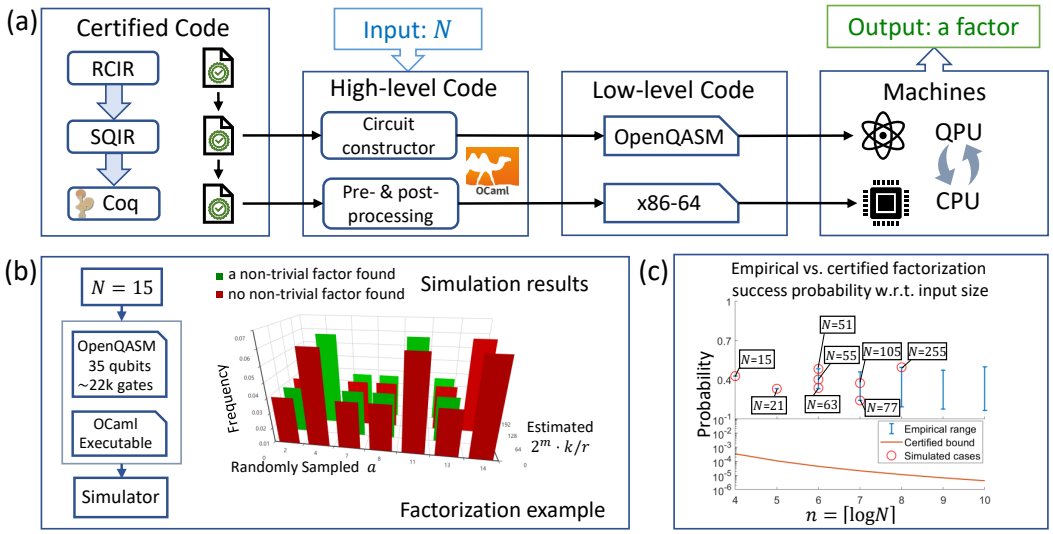


Figure 7.5: End-to-end execution of our implementation of Shor's algorithm. (a) A schematic illustration of the end-to-end quantum-classical hybrid execution. (b) An example of end-to-end execution and simulation of factorization. (c) Empirical statistics (i.e., minimal to maximal success probability) of the success probability of factorization for every valid input N with respect to input size n from 4 to 10 bits.

non-constructive [168] meaning that it makes use of axioms like the law of the excluded middle, whereas one needs to provide constructive proofs [169] in Coq and other proof assistants.

We provide a new, constructive proof without using primitive roots by resorting to the quadratic residues in modulus p^k and connecting whether a randomly chosen a leads to a non-trivial factor to the number of quadratic residues and non-residues in modulus p^k. The counting of the latter is established based on Euler's criterion for distinguishing between quadratic residues and non-residues modulo p^k in Coq.

Putting it all together, we have proved that our implementation of Shor's algorithm successfully outputs a non-trivial factor with a probability of at least $2e^{-2}/\pi^2\lfloor\log_2(N)\rfloor^4$ for one iteration. Furthermore, we also prove in Coq that its failure probability of t repetitions is upper bounded by $(1-2e^{-2}/\pi^2\lfloor\log_2(N)\rfloor^4)^t$, which boosts the success probability of our implementa-

tion arbitrarily close to 1 after $O(\log^4(N))$ repetitions.

We also certify that the gate count in our implementation of Shor's algorithm using OpenQASM's gate set is upper bounded by $(212n^2 + 975n + 1031)m + 4m + m^2$ in Coq, where n refers to the number of bits representing N and m the number of bits in QPE output. Note further $m, n = O(\log N)$, which leads to an $O(\log^3 N)$ overall asymptotic complexity that matches the original paper.

7.3.1 Certifying order finding

For the hybrid order finding procedure, we verify that the success probability is at least $1/\text{polylog}(N)$. Recall that the quantum part of order finding uses in-place modular multiplication ($IMM(a, N)$) and quantum phase estimation (QPE). The classical part applies continued fraction expansion to the outcome of quantum measurements. Our statement of order finding correctness says:

```
Lemma Shor_OF_correct :
  ∀ (a N : ℕ),
    (1 < a < N) → (gcd a N = 1) →
    ℙ[Shor_OF a N = ord a N] ≥ β/⌊log₂(N)⌋⁴.
```

where $\beta = \frac{4e^{-2}}{\pi^2}$. The probability sums over possible outputs of the quantum circuit and tests if post-processing finds `ord a N`.

Certifying IMM

We have proved that our RCIR implementation of IMM satisfies (7.2.3.1). Therefore, because we have a proved-correct translator from RCIR to SQIR, our SQIR translation of IMM also satisfies this property. In particular, the in-place modular multiplication circuit $IMM(a, N)$ with

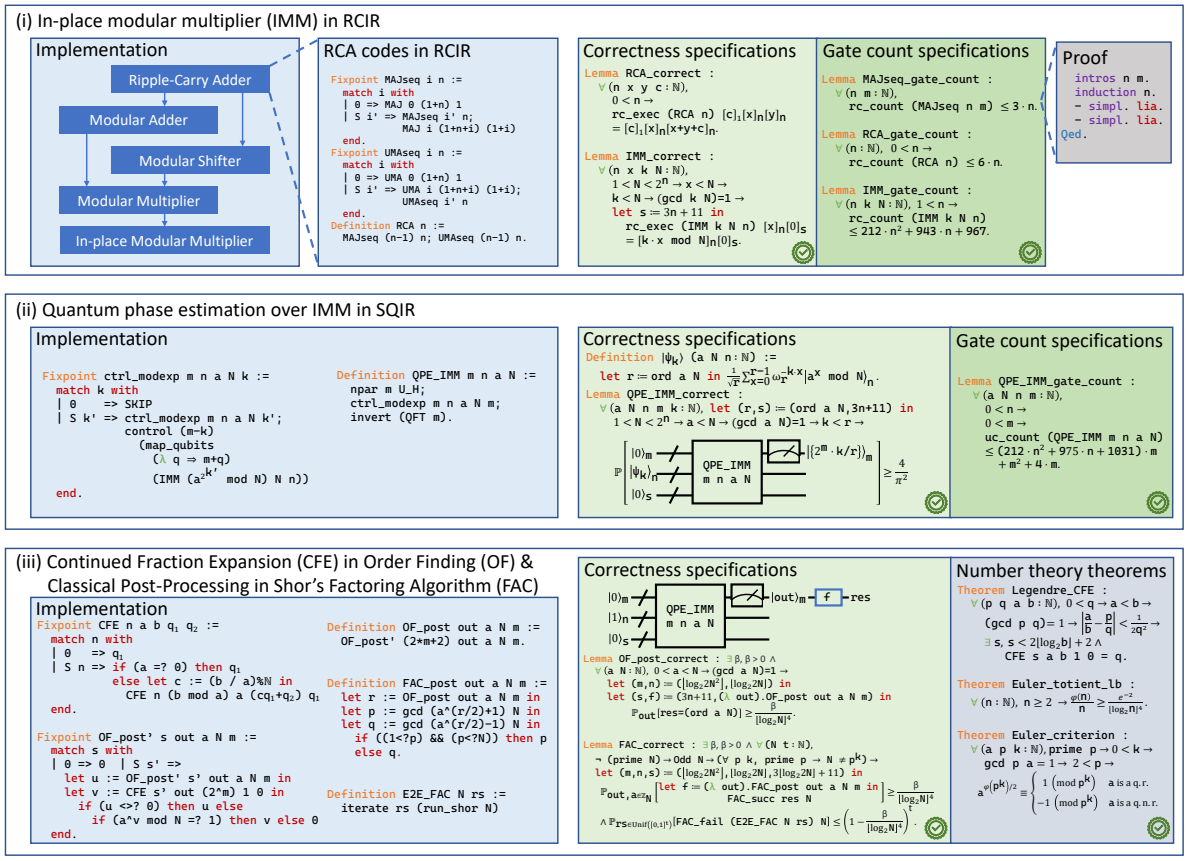


Figure 7.6: Showcases of major components of our end-to-end implementation and corresponding proofs. Codes are adjusted for pretty-printing. (i) The implementation of IMM. We use the example of the Ripple-Carry Adder (RCA) to illustrate the specifications and proofs. (ii) The implementation of quantum phase estimation over IMM in SQIR (QPE_IMM). The correctness specification states that, under some premises, the probability of measuring the closest integer to $2^m k/r$, where r is the order of a modulo N, is larger than a positive constant $4/\pi^2$. We also certify the gate count of the implementation of QPE_IMM. (iii) The implementation of classical post-processing for order finding and factorization. Continued fraction expansion CFE is applied to the outcome of QPE_IMM to recover the order with a certified success probability at least 1/polylog(N). The success probability of factorization is also certified to be at least 1/polylog(N), which can be boosted to 1 minus some exponentially decaying error term after repetitions. These analyses critically rely on number theoretical statements like Legendre's theorem, lower bounds for Euler's totient function, and Euler's criterion for quadratic residues, which have been proven constructively in Coq in our implementation.

n qubits to represent the register and s ancillary qubits, translated from RCIR to SQIR, has the following property for any $0 \leq N < 2^n$ and $a \in \mathbb{Z}_N$:

```
Definition IMMBehavior a N n s c :=
  ∀ x : ℕ, x < N →
    (uc_eval c) × (|x⟩_n ⊗ |0⟩_s) = |a · x mod N⟩_n ⊗ |0⟩_s.
Lemma IMM_correct a N :=
  let n := log2 (2*N) in
  let s := 3*n + 11 in
  IMMBehavior a N n s (IMM a n).
```

Here `IMMBehavior` depicts the desired behavior of an in-place modular multiplier, and we have proved the constructed $IMM(a, N)$ satisfies this property.

Certifying QPE over IMM

We certify that QPE outputs the closest estimate of the eigenvalue's phase corresponding to the input eigenvector with probability no less than $\frac{4}{\pi^2}$:

```
Lemma QPE_semantics :
  ∀ m n z δ (f : ℕ → base_ucom n) (|ψ⟩ : Vector 2^n),
    n > 0 → m > 1 → -1/(2^(m+1)) ≤ δ < 1/(2^(m+1)) →
    Pure_State_Vector |ψ⟩ →
    (∀ k, k < m →
      uc_WT (f k) ∧ (uc_eval (f k)) |ψ⟩ = e^(2^(k+1) π i (z/2^m + δ)) |ψ⟩) →
    ‖⟨z, ψ| (uc_eval (QPE k n f)) |0, ψ⟩‖² ≥ 4/π².
```

To utilize this lemma with $IMM(a, N)$, we first analyze the eigenpairs of $IMM(a, N)$. Let $r = \text{ord}(a, N)$ be the multiplicative order of a modulo N. We define

$$|\psi_j\rangle_n = \frac{1}{\sqrt{r}} \sum_{l<r} \omega_r^{-j \cdot l} |a^l \mod N\rangle_n$$

in SQIR and prove that it is an eigenvector of any circuit satisfying `IMMBehavior`, including

$IMM(a^{2^k}, N)$, with eigenvalue $\omega_r^{j \cdot 2^k}$ for any natural number k, where $\omega_r = e^{\frac{2\pi i}{r}}$ is the r-th primitive root in the complex plane.

```
Lemma IMMBehavior_eigenpair :
  ∀ (a r N j n s k : ℕ) (c : base_ucom (n+s)),
    Order a r N → N < 2^n →
    IMMBehavior a^(2^k) N n s c →
    (uc_eval (f k)) |ψ_j⟩_n ⊗ |0⟩_s = e^(2^(k+1) π i j/r) |ψ_j⟩_n ⊗ |0⟩_s.
```

Here `Order a r N` is a proposition specifying that r is the order of a modulo N. Because we cannot directly prepare $|\psi_j\rangle$, we actually set the eigenvector register in QPE to the state $|1\rangle_n \otimes |0\rangle_s$ using the identity:

```
Lemma sum_of_ψ_is_one :
  ∀ a r N n : ℕ,
    Order a r N → N < 2^n → 1/√r ∑_{k<r} |ψ_j⟩_n = |1⟩_n.
```

By applying `QPE_semantics`, we prove that for any $0 \leq k < r$, with probability no less than $\frac{4}{\pi^2 r}$, the result of measuring QPE applied to $|0\rangle_m \otimes |1\rangle_n \otimes |0\rangle_s$ is the closest integer to $\frac{k}{r} 2^m$.

Certifying Post-processing

Our certification of post-processing is based on two mathematical results (also formally certified in Coq): the lower bound of Euler's totient function and the Legendre's theorem for continued fraction expansion. Let $\mathbb{Z}_n^*$ be the integers smaller than n and coprime to n. For a positive integer n, Euler's totient function $\varphi(n)$ is the size of $\mathbb{Z}_n^*$. They are formulated in Coq as follows.

```
Theorem Euler_totient_lb : ∀ n, n ≥ 2 → φ(n)/n ≥ e^(-2)/⌊log_2 n⌋^4.
Lemma Legendre_CFE :
  ∀ a b p q : ℕ,
    a < b → gcd p q = 1 → 0 < q → |a/b − p/q| < 1/(2q^2) →
    ∃ s, s ≤ 2 log_2(b) + 1 ∧ CFE s a b = q.
```

The verification of these theorems is discussed later.

By Legendre's theorem for CFE, there exists a $s \leq 2m+1$ such that `CFE s out` $2^{\mathtt{m}} = r$, where `out` is the closest integer to $\frac{k}{r}2^m$ for any $k \in \mathbb{Z}_r^*$. Hence the probability of obtaining the order (r) is the sum $\sum_{k \in \mathbb{Z}_r^*} \frac{4}{\pi^2 r}$. Note that $r \leq \varphi(N) < N$. With the lower bound on Euler's totient function, we obtain a lower bound of $1/\text{polylog}(N)$ of successfully obtaining the order $r = \text{ord}(a, N)$ through the hybrid algorithm, finishing the proof of `Shor_OF_correct`.

Lower Bound of Euler's Totient Function

We build our proof on the formalization of Euler's product formula and Euler's theorem by de Rauglaudre [170]. By rewriting Euler's product formula into exponents, we can scale the formula into exponents of Harmonic sequence $\sum_{0<i\leq n} \frac{1}{i}$. Then an upper bound for the Harmonic sequence suffices for the result.

In fact, a tighter lower bound of Euler's totient function exists [171], but obtaining it involves evolved mathematical techniques which are hard to formalize in Coq since they involved analytic number theory. Fortunately, the formula certified above is sufficient to obtain a success probability of at least $1/\text{polylog}(N)$ for factorizing N.

Legendre's Theorem for Continued Fraction Expansion

The proof of Legendre's theorem consists of facts: (1) `CFE s a b` monotonically increases, and reaches b within $2\log_2(b)+1$ steps, and (2) for `CFE s a b` $\leq$ `q` $<$ `CFE (s+1) a b` satisfying $\left|\frac{\mathtt{a}}{\mathtt{b}} - \frac{\mathtt{p}}{\mathtt{q}}\right| < \frac{1}{2\mathtt{q}^2}$, the only possible value for `q` is `CFE s a b`. These are certified following basic analysis to the continued fraction expansion[168].

7.3.2 Certifying Shor's reduction

We formally certify that for half of the possible choices of a, `ord a n` can be used to find a nontrivial factor of N:

```
Lemma reduction_fact_OF :
  ∀ (p k q N : ℕ),
    k > 0 → prime p → 2 < p → 2 < q →
    gcd p q = 1 → N = p^k * q →
    |ℤ_N| ≤ 2 · ∑_{a∈ℤ_N} [1 < gcd (a^⌊ord a N / 2⌋ ± 1) N < N].
```

The expression $[1 < (\texttt{gcd}\ (\texttt{a}^{\lfloor \frac{\texttt{ord a N}}{2} \rfloor} \pm 1)\ \texttt{N}) < \texttt{N}]$ equals to 1 if at least one of $\texttt{gcd}(\texttt{a}^{\lfloor \frac{\texttt{ord a N}}{2} \rfloor} + 1,\ \texttt{N})$ or $\texttt{gcd}(\texttt{a}^{\lfloor \frac{\texttt{ord a N}}{2} \rfloor} - 1,\ \texttt{N})$ is a nontrivial factor of N, otherwise it equals to 0. In the following we illustrate how we achieve this lemma.

From 2-adic Order to Non-Trivial Factors

The proof proceeds as follows: Let $d(x)$ be the largest integer i such that 2^i is a factor of x, which is also known as the 2-adic order. We first certify that $d(\text{ord}(a, p^k)) \neq d(\text{ord}(a, q))$ indicates $a^{\lfloor \frac{\text{ord}(a,N)}{2} \rfloor} \not\equiv \pm 1 \pmod{N}$

```
Lemma d_neq_sufficient :
  ∀ a p q N,
    2 < p → 2 < q → gcd p q = 1 → N = pq →
    d (ord a p) ≠ d (ord a q) →
    a^⌊ord a N / 2⌋ ≢ ±1 (mod N).
```

This condition is sufficient to get a nontrivial factor of N by Euler's theorem and the following lemma

```
Lemma sqr1_not_pm1 :
  ∀ x N,
    1 < N → x^2 ≡ 1 (mod N) → x ≢ ±1 (mod N) →
    1 < gcd (x - 1) N < N ∨ 1 < gcd (x + 1) N < N.
```

By the Chinese remainder theorem, randomly picking a in $\mathbb{Z}_N$ is equivalent to randomly picking b in $\mathbb{Z}_{p^k}$ and randomly picking c in $\mathbb{Z}_q$. $a \equiv b \mod p^k$ and $a \equiv c \mod q$, so $\text{ord}(a, p^k) = \text{ord}(b, p^k)$ and $\text{ord}(a, q) = \text{ord}(c, q)$. Because the random pick of b is independent from the random pick of c, it suffices to show that for any integer i, at least half of the elements in $\mathbb{Z}_{p^k}$ satisfy $d(\text{ord}(x, p^k)) \neq i$.

Detouring to Quadratic Residue

Shor's original proof[2] of this property made use of the existence of a group generator of $\mathbb{Z}_{p^k}$, also known as primitive roots, for odd prime p. But the existence of primitive roots is non-constructive, hence hard to present in Coq. We manage to detour from primitive roots to quadratic residues in modulus p^k in order to avoid non-constructive proofs.

A quadratic residue modulo p^k is a natural number $a \in \mathbb{Z}_{p^k}$ such that there exists an integer x with $x^2 \equiv a \mod p^k$. We observe that a quadratic residue $a \in \mathbb{Z}_{p^k}$ will have $d(\text{ord}(x, p^k)) < d(\varphi(p^k))$, where φ is the Euler's totient function. Conversely, a quadratic non-residue $a \in \mathbb{Z}_{p^k}$ will have $d(\text{ord}(x, p^k)) = d(\varphi(p^k))$:

```
Lemma qr_d_lt :
  ∀ a p k,
    k ≠ 0 → prime p → 2 < p →
    (∃ x, x² ≡ a  mod pᵏ) →
    d (ord a pᵏ) < d (φ (pᵏ)).
Lemma qnr_d_eq :
  ∀ a p k,
    k ≠ 0 → prime p → 2 < p →
    (∀ x, x² ≢ a  mod pᵏ) →
    d (ord a pᵏ) = d (φ (pᵏ)).
```

These lemmas are obtained via Euler's Criterion, which describes the difference between multiplicative orders of quadratic residues and quadratic non-residues. The detailed discussion is put

later.

We claim that the number of quadratic residues in $\mathbb{Z}_{p^k}$ equals to the number of quadratic non-residues in $\mathbb{Z}_{p^k}$, whose detailed verification is left later. Then no matter what i is, at least half of the elements in $\mathbb{Z}_{p^k}$ satisfy $d(\text{ord}(x, p^k)) \neq i$. This makes the probability of finding an $a \in \mathbb{Z}_{p^k q}$ satisfying $d(\text{ord}(a, p^k)) \neq d(\text{ord}(a, q))$ at least one half, in which case one of $\texttt{gcd}\ \left(\texttt{a}^{\lfloor \frac{\texttt{ord a N}}{2} \rfloor} \pm 1\right)\ \texttt{N}$ is a nontrivial factor of N.

Euler's Criterion

We formalize a generalized version of Euler's criterion: for odd prime p and $k > 0$, whether an integer $a \in \mathbb{Z}_{p^k}$ is a quadratic residue modulo p^k is determined by the value of $a^{\frac{\varphi(p^k)}{2}} \mod p^k$.

```
Lemma Euler_criterion_qr :
  ∀ a p k,
    k ≠ 0 → prime p → 2 < p → gcd a p = 1 →
    (∃ x, x² ≡ a  mod pᵏ) →
    a^(φ(pᵏ)/2)  mod pᵏ = 1.
Lemma Euler_criterion_qnr :
  ∀ a p k,
    k ≠ 0 → prime p → 2 < p → gcd a p = 1 →
    (∀ x, x² ≢ a  mod pᵏ) →
    a^(φ(pᵏ)/2)  mod pᵏ = pᵏ - 1.
```

These formulae can be proved by a pairing function over $\mathbb{Z}_{p^k}$:

$$x \mapsto (a \cdot x^{-1}) \mod p^k,$$

where x^{-1} is the multiplicative inverse of x modulo p^k. For a quadratic residue a, only the two solutions of $x^2 \equiv a \mod p^k$ do not form pairing: each of them maps to itself. For each pair (x, y) there is $x \cdot y \equiv a \mod p^k$, so reordering the product $\prod_{x \in \mathbb{Z}_{p^k}} x$ with this pairing proves the

Euler's criterion.

With Euler's criterion, we can reason about the 2-adic order of multiplicative orders for quadratic residues and quadratic non-residues, due to the definition of multiplicative order and $ord(a, p^k)|\varphi(p^k)$.

Counting Quadratic Residues Modulo p^k

For odd prime p and $k > 0$, there are exactly $\varphi(p^k)/2$ quadratic residues modulo p^k in $\mathbb{Z}_{p^k}$, and exactly $\varphi(p^k)/2$ quadratic non-residues.

```
Lemma qr_half :
  ∀ p k,
    k ≠ 0 → prime p → 2 < p →
    |ℤ_{p^k}| = 2 · ∑_{a∈ℤ_{p^k}} [∃x, x² ≡ a  mod p^k].
Lemma qnr_half :
  ∀ p k,
    k ≠ 0 → prime p → 2 < p →
    |ℤ_{p^k}| = 2 · ∑_{a∈ℤ_{p^k}} [∀x, x² ≢ a  mod p^k].
```

Here $[\exists x, x^2 \equiv a \mod p^k]$ equals to 1 if a is a quadratic residue modulo $\mathrm{p^k}$, otherwise it equals to 0. Similarly, $[\forall x, x^2 \not\equiv a \mod \mathrm{p^k}]$ represents whether a is a quadratic non-residue modulo $\mathrm{p^k}$. These lemmas are proved by the fact that a quadratic residue a has exactly two solutions in $\mathbb{Z}_{p^k}$ to the equation $x^2 \equiv a \mod p^k$. Thus for the two-to-one self-map over $\mathbb{Z}_{p^k}$

$$x \mapsto x^2 \bmod p^k,$$

the size of its image is exactly half of the size of $\mathbb{Z}_{p^k}$. To prove this result in Coq, we generalize two-to-one functions with mask functions of type $\mathbb{N} \to \mathbb{B}$ to encode the available positions, then reason by induction.

7.3.3 End-to-end certification

We present the final statement of the correctness of the end-to-end implementation of Shor's algorithm.

```
Theorem end_to_end_shor_fails_with_low_probability :
  ∀ N niter,
    ¬ (prime N) → Odd N →
    (∀ p k, prime p → N ≠ p^k) →
    ℙ_{rnds∈Uniform([0,1]^niter)}[end_to_end_shor N rnds = None]
      ≤ (1 - (1/2) * (β / (log2 N)^4))^niter.
```

Then r can be less than an arbitrarily small positive constant ϵ by enlarging `niter` to $\frac{2}{\beta} \ln \frac{1}{\epsilon} \log_2^4 N$, which is $O(\log^4 N)$.

This theorem can be proved by combining the success probability of finding the multiplicative order and the success probability of choosing proper a in the reduction from factorization to order finding. We build an ad-hoc framework for reasoning about discrete probability procedures to express the probability here.

7.3.4 Certifying resource bounds

We provide a concrete polynomial upper bound on resource consumption in our implementation of Shor's algorithm. The aspects of resource consumption considered here are the number of qubits and the number of primitive gates supported by OpenQASM 2.0 [172]. The number of qubits is easily bounded by the maximal index used in the SQIR program, which is linear to the length of the input. For gate count bounds, we reason about the structure of our circuits. We first generate the gate count bound for the RCIR program, then we transfer this bound to the bound for the SQIR program. Eventually, the resource bound is given by

```
Lemma ugcount_shor_circuit :
  ∀ a N,
    0 < N →
    let m := Nat.log2 (2*(N^2)) in
    let n := Nat.log2 (2*N) in
    ugcount (shor_circuit a N) ≤
    (212*n*n + 975*n + 1031)*m + 4*m + m*m.
```

Here `ugcount` counts how many gates are in the circuit. Note $m, n = O(\log N)$. This gives the gate count bound for one iteration as $(212n^2+975n+1031)m+4m+m^2 = O(\log^3 N)$, which is asymptotically the same as the original paper [2], and similar to other implementations of Shor's algorithm [166, 167] (up to $O(\log\log N)$ multiplicative difference because of the different gate sets).

Using the certified bound, we may estimate the upper bound of gates in large factorization circuits in our implementation. To factorize a number of 1024-bit, our implementation will generate circuits with at most 4.58×10^{11} gates. This upper bound is several orders of magnitudes larger than the estimation in [166, 167] (orders of magnitudes around 10^9) because our implementation is not optimized to reduce the total gate count but for a structured certification procedure.

7.4 Running certified code

The codes are certified in Coq, which is a language designed for formal verification. To run the codes realistically and efficiently, extractions to other languages are necessary. Our certification contains the quantum part and the classical part. The quantum part is implemented in SQIR embedded in Coq, and we extract the quantum circuit into OpenQASM 2.0 [172] format. The

classical part is extracted into OCaml code following Coq's extraction mechanism [173]. Then the OpenQASM codes can be sent to a quantum computer (in our case, a classical simulation of a quantum computer), and OCaml codes are executed on a classical computer.

With a certification of Shor's algorithm implemented inside Coq, the guarantees of correctness on the extracted codes are strong. However, although our Coq implementation of Shor's algorithm is fully certified, extraction introduces some trusted code outside the scope of our proofs. In particular, we trust that extraction produces OCaml code consistent with our Coq definitions and that we do not introduce errors in our conversion from SQIR to OpenQASM. We "tested" our extraction process by generating order-finding circuits for various sizes and confirming that they produce the expected results in a simulator.

7.4.1 Extraction

For the quantum part, we extract the Coq program generating SQIR circuits into the OCaml program generating the corresponding OpenQASM 2.0 assembly file. We substitute the OpenQASM 2.0 gate set for the basic gate set in SQIR, which is extended with: $X, H, U_1, U_2, U_3, CU_1$, $SWAP, CSWAP, CX, CCX, C3X, C4X$. Here X, H are the Pauli X gate and Hadamard gate. U_1, U_2, U_3 are single-qubit rotation gates with different parametrization [172]. CU_1 is the controlled version of the U_1 gate. $SWAP$ and $CSWAP$ are the swap gate and its controlled version. $CX, CCX, C3X$, and $C4X$ are the controlled versions of the X gate, with a different number of control qubits. Specifically, CX is the CNOT gate. The proofs are adapted with this gate set. The translation from SQIR to OpenQASM then is direct.

For the classical part, we follow Coq's extraction mechanism. We extract the integer types

in Coq's proof to OCaml's `Z` type, and several number theory functions to their correspondence in OCaml with the same behavior but better efficiency. Since our proofs are for programs with classical probabilistic procedures and quantum procedures, we extract the sampling procedures with OCaml's built-in randomization library.

One potential gap in our extraction of Coq to OCaml is the assumption that OCaml floats satisfy the same properties as Coq Real numbers. It is actually not the case, but we did not observe any error introduced by this assumption in our testing. In our development, we use Coq's axiomatized representation of reals [174], which cannot be directly extracted to OCaml. We chose to extract it to the most similar native data type in OCaml–floating-point numbers. An alternative would be to prove Shor's algorithm correct with gate parameters represented using some Coq formalism for floating-point numbers [175], which we leave for future work.

7.4.2 Experiments

We test the extracted codes by running small examples on them. Since nowadays quantum computers are still not capable of running quantum circuits as large as generated Shor's factorization circuits ($\sim$30 qubits, $\sim 10^4$ gates for small cases), we run the circuits with the DDSIM simulator [176] on a laptop with an Intel Core i7-8705G CPU. The experiment results are included in Figure 7.5 (b) (c).

As a simple illustration, we showcase the order finding for $a = 3$ and $N = 7$ on the left of Figure 7.5 (b). The extracted OpenQASM file makes use of 29 qubits and contains around 11000 gates. DDSIM simulator executes the file and generates simulated outcomes for 10^5 shots. The measurement results of QPE are interpreted in binary representation as estimated $2^m \cdot k/r$.

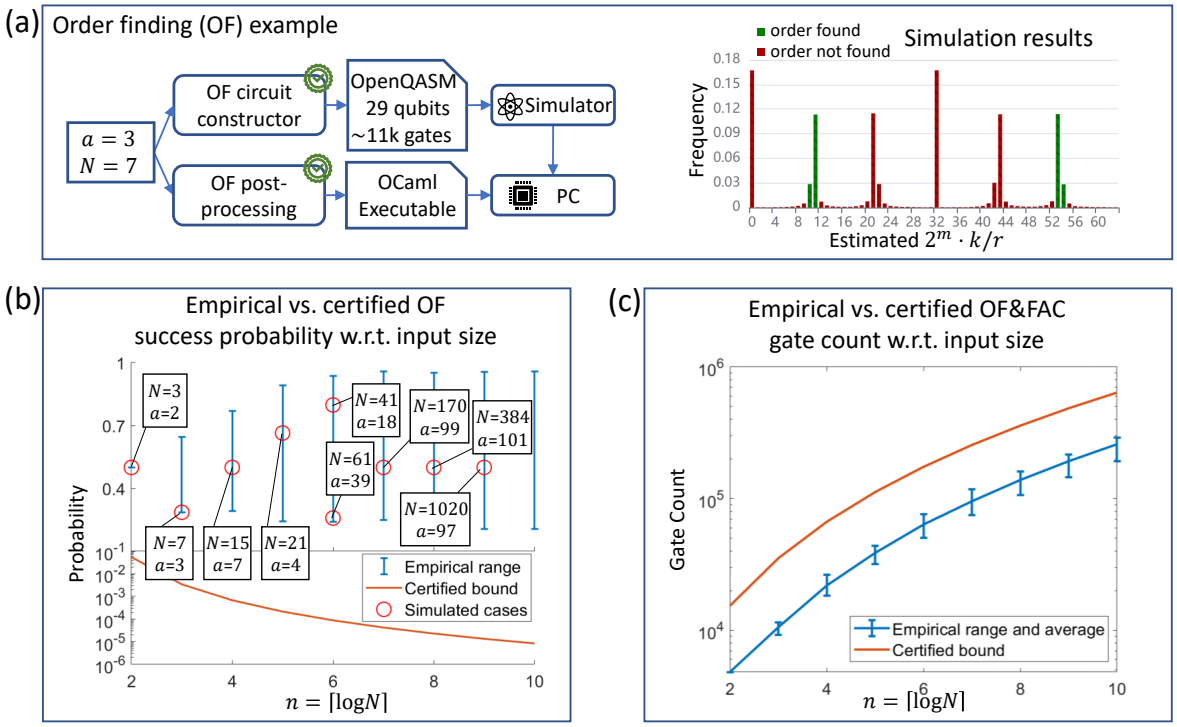


Figure 7.7: End-to-end execution of the order-finding algorithm. (a) Examples of end-to-end executions of order finding (OF). The left example finds the order for a=3 and N=7. The generated OpenQASM file uses 29 qubits and contains around 11k gates. We employed JKQ DDSIM [176] to simulate the circuit for 100k shots, and the frequency distribution is presented. The trials with post-processing leading to the correct order r=6 are marked green. The empirical success probability is 28.40%, whereas the proved success probability lower bound is 0.34%. (b, c) Empirical statistics of the gate count and success probability of order finding for every valid input N with respect to input size n from 2 to 10 bits. We draw the bounds certified in Coq as red curves. Whenever the simulation is possible with DDSIM, we draw the empirical bounds as red circles. Otherwise, we compute the corresponding bounds using analytical formulas with concrete inputs. These bounds are drawn as blue intervals called empirical ranges (i.e., minimal to maximal success probability) for each input size.

In this case, the outcome ranges from 0 to 63, with different frequencies. We apply OCaml post-processing codes for order finding on each outcome to find the order. Those measurement outcomes reporting the correct order (which is 6) are marked green in Figure 7.5 (b). The frequency summation of these measurement outcomes over the total is 28.40%, above the proven lower bound of the success probability of order finding which is 0.17% for this input.

We are also able to simulate the factorization algorithm for $N = 15$. For any a coprime to 15, the extracted OpenQASM codes contain around 35 qubits and 22000 gates. Fortunately, DDSIM still works efficiently on these cases due to the well-structured states of these cases, taking around 10 seconds for each simulation. We take 7×10^5 shots in total. When $N = 15$, the measurement outcomes from QPE in order finding are limited to $0, 64, 128, 192$ because the order of any a coprime to 15 is either 2 or 4, so $2^m \cdot k/r$ can be precisely expressed as one of them without approximation. The frequency of the simulation outcomes for $N = 15$ is displayed on the right of Figure 7.5 (b). We then apply the extracted OCaml post-processing codes for factorization to obtain a non-trivial factor of N. The overall empirical success probability is 43.77%, above our certified lower bound of 0.17%.

We have also tested larger cases on DDSIM simulator [176] for input size ranging from 2 bits to 10 bits (correspondingly, N from 3 to 1023), as in Figure 7.5 (c). Since the circuits generated are large, most of the circuits cannot be simulated in a reasonable amount of time (we set the termination threshold 1 hour). We exhibit selected cases that DDSIM is capable of simulating: $N = 15, 21, 51, 55, 63, 77, 105, 255$ for factorization, and $(a, N) = (2, 3), (3, 7), (7, 15), (4, 21), (18, 41), (39, 61), (99, 170), (101, 384), (97, 1020)$ for order finding. These empirically investigated cases are drawn as red circles in Figure 7.5 (c). Most larger circuits that are simulated by DDSIM have the multiplicative order a power of 2 so that the simulated state is efficiently

expressible. For each input size, we also calculate the success probability for each possible input combination by using the analytical formulae of the success probability with concrete inputs. Shor shows the probability of obtaining a specific output for order finding is[2]

$$\mathbb{P}[\texttt{out} = u] = \frac{1}{2^{2m}} \sum_{0 \le k < r} \left| \sum_{\substack{0 \le v < r \\ v \equiv k \pmod r}} e^{2\pi i u v / 2^m} \right|^2 .$$

Here r is the order and m is the precision used in QPE. The success probability of order finding then is a summation of us for which the post-processing gives correct r. For most output u, the probability is negligible. The output tends to be around $2^m k/r$, so the sum is taken over integers whose distance to the closest $2^m k/r$ (for some k) is less than a threshold, and the overall probability of getting these integers is at least 95%. Hence the additive error is less than 0.05. These empirical results are drawn as blue intervals (i.e., minimal to maximal success probability) in Figure 7.5 for each input size, which is called the empirical range of success probability. The certified probability lower bounds are drawn as red curves in Figure 7.5 as well. The empirical bounds are significantly larger than the certified bounds for small input sizes because of loose scaling in proofs, and non-optimality in our certification of Euler's totient function's lower bounds. Nevertheless, asymptotically our certified lower bound is sufficient for showing that Shor's algorithm succeeds in polynomial time with large probability.

We also exhibit the empirical gate count and certified gate count for order finding and factorization circuits. The circuits for order finding are exactly the factorization circuits after a is picked, so we do not distinguish these two problems for gate count. On the right of Figure 7.5 (c), we exhibit these data for input sizes ranging from 2 to 10. We enumerate all the inputs for

these cases and calculate the maximal, minimal, and average gate count and draw them as blue curves and intervals. The certified gate count only depends on the input size, which is drawn in red. One can see the empirical results satisfy the certified bounds on gate count. Due to some scaling factors in the analytical gate count analysis, the certified bounds are relatively loose. Asymptotically, our certified gate count is the same as the original paper's analysis.

Chapter 8: Algebraic Reasoning of Quantum While Programs

Chapter summary

Compilers serve as an essential role in the quantum software stacks to reduce resource consumption of running quantum programs, while their correctness needs high-assurance: the rewrites conducted by compilers should keep the behaviors of programs the same. In this chapter, we investigate the *algebraic* reasoning of quantum programs.

The theory in this chapter is inspired by the success of classical program analysis based on Kleene algebra. One prominent example of such is the famous Kleene Algebra with Tests (KAT), which has furnished both theoretical insights and practical tools. The succinctness of algebraic reasoning would be especially desirable for scalable analysis of quantum programs, given the involvement of exponential-size matrices in most of the existing methods. A few key features of KAT including the idempotent law and the nice properties of classical tests, however, fail to hold in the context of quantum programs due to their unique quantum features, especially in branching. We propose Non-idempotent Kleene Algebra (NKA) as a natural alternative and identify complete and sound semantic models for NKA as well as their quantum interpretations. In light of applications of KAT, we demonstrate algebraic proofs in NKA of quantum compiler optimization and the normal form of quantum **while**-programs. Moreover, we extend NKA with Tests (i.e., NKAT), where tests model quantum predicates following effect algebra, and illustrate

how to encode propositional quantum Hoare logic as NKAT theorems.

8.1 Introduction

8.1.1 Background and motivation

Kleene algebra (KA) [177] that establishes the equivalence of regular expressions and finite automata is an important connection built between programming languages and abstract machines with a wide range of applications. One very successful extension of KA, called Kleene algebra with tests (KAT), was introduced by Kozen [8] that combines KA with Boolean algebra (BA) to model the fundamental constructs arising in programs: sequencing, branching, iteration, etc. More importantly, the equational theory of KAT, which can be finitely axiomatized [178], allows *algebraic reasoning* about corresponding classical programs.

The mathematical elegance and succinctness of algebraic reasoning with KAT have furnished deep theoretical insights as well as practical tools. A lot of topics can be investigated with KAT including, e.g., program transformations [7], compiler optimization [179], Hoare logic [180], and so on. An important recent application of KAT is NetKAT [181] that reasons about the packet-forwarding behavior of software-defined networks, with both a solid theoretical foundation [182] and scalable practical performance [181]. An efficient fragment of KAT, called Guarded KAT (GKAT), has also been identified [183] to model typical imperative programs with an almost linear time equational theory. In contrast, KAT's equational theory is **PSPACE**-complete [184].

Quantum computation has been a topic of significant recent interest. With breakthroughs in experimental quantum computing and the introduction of many quantum programming languages

such as Quipper [16], Scaffold [185], QWIRE [186], Microsoft's Q# [187], IBM's Qiskit [188], Google's Cirq [189], Rigetti's Forest [190], there is an imperative need for the analysis and verification of quantum programs.

Indeed, program analysis and verification have been a central topic ever since the seminal work on quantum programming languages [191, 192, 193, 194, 195]. There have been many attempts of developing Hoare-like logic [196] for verification of quantum programs [197, 198, 199, 200, 201, 202]. In particular, D'Hondt and Panangaden [203] proposed the notions of quantum predicate and weakest precondition. Ying [202] established the quantum Hoare logic with (relative) completeness for reasoning about a quantum extension of the **while**-language with many subsequent developments [204, 205, 206]. We refer curious readers to surveys [207, 208, 209] for details.

Quantum **while**-programs have similar (yet semantically different) fundamental constructs (e.g., sequencing, branching, iterations) like classical ones, which gives rise to a natural question of the possibility of using KA/KAT to algebraically reason about quantum programs. Existing methods for quantum program analysis and verification usually involve exponential-size matrices in terms of the system size, which hence significantly limits the scalability. In contrast, a succinct KA-based algebraic reasoning, if possible, would greatly increase the scalability of such analyses for quantum programs due to its mathematical succinctness.

8.1.2 Research challenges and solutions

Let us first revisit KAT-based algebraic reasoning and highlight the challenges in extending the framework to the quantum setting. We assume a few self-explanatory quantum notations with

detailed quantum preliminaries in Section 8.3.1.

KAT-based Reasoning. A typical reasoning framework based on KAT, similarly for NetKAT and GKAT, will establish that KAT models the targeted computation by showing

$$\vdash_{\mathrm{KAT}} e = f \quad \Leftrightarrow \quad \forall \mathrm{int}, \mathcal{K}_{\mathrm{int}}(e) = \mathcal{K}_{\mathrm{int}}(f), \tag{8.1.2.1}$$

where $\mathcal{K}_{\mathrm{int}}$ is an interpretation mapping from expressions to a language (or semantic) model of the desired computation. In reasoning about while programs, one encodes them as KAT expressions as in Propositional Dynamic Logic [210]:

$$p; q := pq \tag{8.1.2.2}$$

$$\textbf{if } b \textbf{ then } p \textbf{ else } q := bp + \overline{b}q \tag{8.1.2.3}$$

$$\textbf{while } b \textbf{ do } p \textbf{ done} := (bp)^*\overline{b}, \tag{8.1.2.4}$$

where b is a classical guard/test and $\overline{b}$ is its Boolean negation.

Intuitively, if one can derive the equivalence of encodings of two classical programs in KAT, then through the soundness direction ($\Rightarrow$), one can also establish the equivalence between the semantics of the original programs by applying an appropriate interpretation.

Quantum Branching. One *critical* difference between quantum and classical programs lies in the *branching* statement. The quantum branching statement,

$$\textbf{case } M[q] \rightarrow^i P_i \textbf{ end}, \tag{8.1.2.5}$$

refers to a *probabilistic* procedure to execute branch P_i depending on the outcome of quantum measurement M on quantum variable q (of which the state is denoted by a density operator ρ). Consider the two-branching case (i=0,1), and let $M = \{M_0, M_1\}$ be the quantum measurement operators. Measurement M will *collapse* ρ to the state $\rho_0 = M_0\rho M_0^\dagger / \operatorname{Tr}\left(M_0\rho M_0^\dagger\right)$ with probability $p_0 = \operatorname{Tr}\left(M_0\rho M_0^\dagger\right)$, and the state $\rho_1 = M_1\rho M_1^\dagger / \operatorname{Tr}\left(M_1\rho M_1^\dagger\right)$ with probability $p_1 = \operatorname{Tr}\left(M_1\rho M_1^\dagger\right)$ respectively (here $\operatorname{Tr}(\cdot)$ is the matrix trace). After the measurement M, the program will execute P_i on state ρ_i with probability p_i $(i = 0, 1)$.

There are two important differences between quantum and classical branching. The *first* is that quantum branching allows probabilistic choices over different branches. Even though random choices also appear in probabilistic programs, the probabilistic choices in quantum branching are due to quantum mechanics (i.e., measurements). In particular, their distributions are determined by the underlying quantum states and the corresponding quantum measurements, and hence *implicit* in the syntax of quantum programs, whereas specific probabilities are usually *explicitly* encoded in the syntax of probabilistic programs. Moreover, different quantum measurements do not necessarily commute with each other, which could hence lead to more complex probability distributions in quantum branching than ones allowed in classical probability theory and hence probabilistic programs.

The *second* difference lies in the different roles played by classical guards and quantum measurements in branching. Note that classical guards serve two functionalities simultaneously: (1) first, their values are used to choose the branches before the control; (2) second, they can also be deemed as property tests (i.e. logical propositions) on the state of the program after the control but before executing each branch. These two points might be so natural that one tends to forget that they are based on *an assumption that observing the guard won't change the state*

of the program, which is also naturally held classically. The classical guards, when deemed as tests in KAT, enjoy further the Boolean algebraic properties so that they can be conveniently manipulated.

This natural assumption, however, fails to hold in quantum branching since quantum measurements will change underlying states in the branching statement. This is mathematically evident as we see ρ is collapsed to either ρ_0 or ρ_1 for different branches. Therefore, it is conceivable that quantum branching (and hence quantum programs) should refer to a different semantic model and quantum measurements should be deemed different from the tests in KAT.

Issues with directly adopting KAT/KA. Aforementioned differences make it hard to directly work with KAT/KA for quantum programs. First, there is a well-known issue when combining non-determinism, which is native to KAT, with probabilistic choices [211, 212], the latter of which is however essential in quantum branching. A similar issue also showed up in the probabilistic extension of NetKAT [213], which does not satisfy all the KAT rules, especially the *idempotent* law. One might wonder about the possibility of using GKAT [183], which is designed to mitigate this issue by restricting KAT with guarded structures. Unfortunately, the classical guarded structure modeled in GKAT is semantically different from quantum branching, which makes it hard to connect GKAT with appropriate quantum models.

Solution with NKA and NKAT. Our strategy is to work with the variant of KA without the idempotent law, namely, the *non-idempotent Kleene algebra* (NKA). This change will help model the probabilistic nature of quantum programs in a natural way, however, at the cost of losing properties implied by the idempotent law. Fortunately, thanks to the existing research on NKA [214, 215], many properties of KA are recovered in NKA for its applications to quantum pro-

grams.

Since there is no single ”test” in quantum programs that can serve two purposes like classical guards, we simply separate the treatments for them. The branching functionality of quantum measurements can hence be expressed in NKA by treating them as normal program statements. Precisely, any quantum two-branching can be encoded as

$$m_0 p_0 + m_1 p_1, \tag{8.1.2.6}$$

where $m_{0/1}$ are encodings of measurements and $p_{0/1}$ are encodings of programs in each branch. Comparing with the classical encoding (8.1.2.3), $m_{0/1}$ no longer enjoy the Boolean algebraic properties and should be treated separately.

It turns out that many classical applications of KAT such as compiler optimization [179] and the proof of the normal form of **while**-programs [8] can be implemented in NKA for quantum programs with branching functionality only.

However, one needs to extend NKA to recover other applications of KAT which makes essential use of the proposition functionality of tests. A prominent example in KAT is its application to propositional Hoare logic [180]. Indeed, a typical Hoare triple $\{b\}p\{c\}$ asserts that whenever b holds before the execution of the program p, then if and when p halts, c will hold of the output state, where b, c are both tests in KAT leveraging their proposition functionality.

A similar triple $\{A\}P\{B\}$ is also used in quantum Hoare logic [202], where P is the quantum program and A, B become *quantum predicates* [203]. To encode quantum Hoare logic, we extend NKA with the ”test”, denoted NKAT, which mimics the behavior of quantum predicates following the effect algebra [216]. With quantum predicates, we develop a more delicate descrip-

tion of measurements in quantum branching, called *partitions*, which allow us to reason about the relationship among quantum branches caused by the same quantum measurement, e.g., the m_0 and m_1 branches in (8.1.2.6).

Quantum Path Model. One of our main technical contributions is the identification of the so-called *quantum path model*, a complete and sound semantic model for NKA. Namely,

$$\vdash_{\mathrm{NKA}} e = f \quad \Leftrightarrow \quad \forall \mathrm{int}, \mathcal{Q}_{\mathrm{int}}(e) = \mathcal{Q}_{\mathrm{int}}(f), \tag{8.1.2.7}$$

where $\mathcal{Q}_{\mathrm{int}}$ is an interpretation mapping from NKA expressions to quantum path actions, which can be deemed as quantum evolution in the path integral formulation of quantum mechanics. $\mathcal{Q}_{\mathrm{int}}$ will connect the NKA encoding of any quantum program P with its denotational semantics $[\![P]\!]$. Since we relate NKA to quantum models which imply the probabilistic feature inherently, there is no need to explicitly add probability to NKA.

The key motivation of the quantum path model is to address the infinity issue in NKA. For an intuitive understanding, one can deem any KA or NKA expression as a collection of potentially infinitely many traces, where ”infinitely many” is caused by $*$ operations. In the case of KA, by the idempotent law, every single trace is either in or out of the collection. However, in the case of NKA, each trace is associated with a weight, which by itself could be infinite. To distinguish between nonequivalent NKA expressions, one needs to build a semantic model that can characterize a collection of weighted traces with potentially infinite weights. We also require the quantum nature of this semantic model for connection with the denotational semantics of quantum programs.

The path integral formulation becomes very natural in this regard: it formulates quantum

evolution as the accumulative effect of a collection of evolutions on individual trajectories. Our quantum path model basically characterizes the accumulative quantum evolution over a collection of potentially infinite evolutions over individual traces. By identifying quantum path actions representing quantum predicates and quantum measurements in the quantum path model, a soundness theorem is proved for NKAT as well.

Quantum-Classical differences as exhibited in NKA and NKAT. The quantum-classical difference is not explicit in the syntax of NKA, as there is no special symbol for quantum measurements. This is also reflected in the proof of the completeness of NKA where an interpretation of essentially classical probabilistic processes is constructed (Remark 8.4.1).

However, the difference becomes explicit in NKAT: the two functionalities of the quantum guards are characterized separately by *effects* and *partitions*, in contrast with the classical guards in KAT. The general noncommutativity of quantum measurements in NKAT demonstrates its quantumness and distinguishes itself from any classical model.

Main Theorem. Our main theorem presented below formally guarantees that quantum program equivalences are implied if we can algebraically derive the corresponding NKA theorems. This approach is similar to deriving classical program equivalence via KAT.

Theorem 8.1.1. *Given two quantum programs P, Q and sub-program pairs $\{\langle P_i, Q_i\rangle\}$ where $[\![P_i]\!] = [\![Q_i]\!]$, if Horn theorem*

$$\vdash_{\mathrm{NKA}} \left(\bigwedge_i \mathrm{Enc}(P_i) = \mathrm{Enc}(Q_i)\right) \to \mathrm{Enc}(P) = \mathrm{Enc}(Q)$$

is derivable, then we have $[\![P]\!] = [\![Q]\!]$. Here Enc *is the encoding of quantum program in a similar manner of* (8.1.2.2)-(8.1.2.4).

Related Works. It is worthwhile comparing quantum algebraic reasoning based on NKA with other techniques on quantum program analysis, e.g., quantum Hoare logic [202]. As we see, classical algebraic reasoning is extremely good at certain tasks (e.g, equational proofs). However, since it abstracts away a lot of semantic information, it cannot tell about detailed specifications on the state of programs, which can otherwise be reasoned by Hoare logic [196].

Our quantum algebraic reasoning inherits the advantages and disadvantages of its classical counterpart. It allows elegant applications in Section 8.5 & 8.6, which is very hard (e.g., involving exponential-size matrices) to solve with the quantum Hoare logic [202] or its relational variants [217, 218]. However, it cannot replace quantum Hoare logic to reason about, e.g., specifications on the state of quantum programs either.

A recent result of quantum abstract interpretation [219] contributes to another promising approach to verifying quantum assertions with succinct proofs, although its applicability and technique are incomparable to ours.

There are many other verification tools developed for quantum programs. Hietala et al. [220] built VOQC, an infrastructure for quantum circuits in Coq with numerous verified programs and compiler optimization rules. Another theory for equational reasoning of quantum circuits is introduced in [221]. They serve as good complements of our framework when loops are absent.

8.2 Non-idempotent Kleene algebra

In this section, we introduce the theory of a Kleene algebraic system without the idempotent law, which is called non-idempotent Kleene algebra (NKA).

We inherit Kozen's axiomatization for Kleene algebra (KA) in [6] with several weakenings.

Axioms of KA	**Axioms of NKA**
SEMIRING LAWS	SEMIRING LAWS:
$p + (q + r) = (p + q) + r;$	$p + (q + r) = (p + q) + r;$
$p + q = q + p;$	$p + q = q + p;$
$p + 0 = p;$	$p + 0 = p;$
$p(qr) = (pq)r;$	$p(qr) = (pq)r;$
$1p = p1 = p;$	$1p = p1 = p;$
$0p = p0 = 0;$	$0p = p0 = 0;$
$p(q + r) = pq + pr;$	$p(q + r) = pq + pr;$
$(p + q)r = pr + qr;$	$(p + q)r = pr + qr;$
$p + p = p;$	
PARTIAL ORDER LAWS	PARTIAL ORDER LAWS
$p \leq q \leftrightarrow p + q = q;$	$p \leq p;$
	$p \leq q \wedge q \leq p \rightarrow p = q;$
	$p \leq q \wedge q \leq r \rightarrow p \leq r;$
	$p \leq q \wedge r \leq s \rightarrow p + r \leq q + s;$
	$p \leq q \wedge r \leq s \rightarrow pr \leq qs;$
STAR LAWS	STAR LAWS
$1 + pp^* \leq p^*;$	$1 + pp^* \leq p^*;$
$q + pr \leq r \rightarrow p^*q \leq r;$	$q + pr \leq r \rightarrow p^*q \leq r;$
$q + rp \leq r \rightarrow qp^* \leq r;$	$q + rp \leq r \rightarrow qp^* \leq r;$

Figure 8.1: Axioms of KA and NKA. Axioms marked in blue (red) only present in NKA (KA).

Definition 8.2.1. *A non-idempotent Kleene algebra (NKA) is a 7-tuple* $(\mathcal{K}, +, \cdot, *, \leq, 0, 1)$*, where* $+$ *and* $\cdot$ *are binary operations,* $*$ *is a unary operation, and* $\leq$ *is a binary relation. It satisfies the axioms in Figure 8.1.*

The most essential weakening is the deletion of the idempotent law. The partial order in KA cannot directly fit in the scenario when the idempotent law is absent. We hence generalize the KA partial order to any partial order that is preserved by $+$ and $\cdot$. Therefore, $*$ also preserves this partial order. Moreover, we did not include the symmetric fixed point inequality $1 + p^*p \leq p^*$

$1 + pp^* = 1 + p^*p = p^*$	(fixed-point)
$p \leq q \to p^* \leq q^*$	(monotone-star)
$1 + p(qp)^*q = (pq)^*$	(product-star)
$(pq)^*p = p(qp)^*$	(sliding)
$(p+q)^* = (p^*q)^*p^* = p^*(qp^*)^*$	(denesting)
$0 \leq p$	(positivity)

(a) Commonly used theorems of NKA

$(pp)^*(1+p) = p^*$	(unrolling)
$pq = qp \to p^*q = qp^*$	(swap-star)
$pq = rp \to pq^* = r^*p$	(star-rewrite)

(b) Several theorems of NKA for applications

Figure 8.2: Derivable formulae in NKA.

because it is derivable by other axioms, both in KA and in NKA [215].

Definition 8.2.2. *For an alphabet Σ, an expression over Σ is inductively defined by:*

$$e ::= 0 \mid 1 \mid a \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e_1^*, \tag{8.2.0.1}$$

where $a \in \Sigma$. We denote all the expressions over Σ by Exp_Σ.

A Horn formula ϕ is defined as the form $(\bigwedge_i e_i \leq f_i) \to e \leq f$. One may also substitute equation for inequality in ϕ since $e = f \leftrightarrow e \leq f \wedge f \leq e$.

We write $\vdash_{\mathrm{NKA}} \phi$ if ϕ is derivable in NKA with equational logic. Any derivable formula in NKA is a theorem *of NKA.*

Apparently, every theorem in NKA is derivable in KA, since the partial order in KA is monotone. The reverse direction is not true in general. Indeed, the idempotent law, for example, is nowhere derivable from the NKA axioms. It is thus natural to ask what important theorems in KA are still derivable in NKA. We provide affirmative answers to many of them in the following.

Lemma 8.2.3. *The following formulae are derivable in NKA.*

1. *The formulae in Figure 8.2a due to [215].*

2. *The formulae in Figure 8.2b.*

Proof of Lemma 8.2.3. We rewrite the proofs in [215] for the rules in Figure 8.2a.

- $(1 + pp^* = p^*)$: By star laws there is $1 + pp^* \leq p^*$, so we only need to prove the other side.

  Because $\leq$ is monotone, we multiply p and then plus 1 on the both sides, leading to

  $$1 + p(1 + pp^*) \leq 1 + pp^*.$$

  Applying the inductive star law gives $p^* \leq 1 + pp^*$.

- $(1 + p^*p = p^*)$: First we show $\geq$ side. Notice that

  $$1 + p(1 + p^*p) = 1 + p + pp^*p = 1 + (1 + pp^*)p = 1 + p^*p.$$

  Applying the inductive star law, we have $p^* \leq 1 + p^*p$.

  Then we show $\leq$ side. Applying star law,

  $$p + ppp^* = p(1 + pp^*) \leq pp^*.$$

  So $p^*p \leq pp^*$ holds. Because $\leq$ is preserved by $+$, we conclude $1 + p^*p \leq 1 + pp^* \leq p^*$.

- $(p \leq q \rightarrow p^* \leq q^*)$: We multiply q^* and add 1 on both sides, which gives

  $$1 + pq^* \leq 1 + qq^* \leq q^*.$$

  By star laws, there is $p^* \leq q^*$.

- $(1 + p(qp)^*q = (pq)^*)$: We show $\geq$ side first. By semiring laws there is

$$1 + (pq)(1 + p(qp)^*q) = 1 + p(1 + qp(qp)^*)q = 1 + p(qp^*)q.$$

  Because of the inductive star law, we get $(pq)^* \leq 1 + p(qp)^*q$.

  Similarly for $\leq$ side, we consider

$$q + qpq(pq)^* = q(1 + pq(pq)^* = q(pq)^*.$$

  We know that $(qp)^*q \leq q(pq)^*$. Multiplying p and adding 1 on the both sides give

$$1 + p(qp)^*q \leq 1 + pq(pq)^* \leq (pq)^*.$$

- $((pq)^*p = p(qp)^*)$: Multiplying p on product-star results in

$$(pq)^*p = p + p(qp)^*qp = p(qp)^*.$$

- $((p + q)^* = (p^*q)^*p^*)$: To show $(p + q)^* \leq (p^*q)^*p^*$, we apply sliding twice, fixed-point

twice, followed by sliding once :

$$
\begin{aligned}
1 + (p+q)(p^*q)^*p^* &= 1 + p(p^*q)^*p^* + q(p^*q)^*p^* \\
&= pp^*(qp^*)^* + (1 + (qp^*)^*qp^*) \\
&= pp^*(qp^*)^* + (qp^*)^* \\
&= (1 + pp^*)(qp^*)^* \\
&= p^*(qp^*)^* \\
&= (p^*q)^*p^*
\end{aligned}
$$

Then by the inductive star law there is $(p+q)^* \leq (p^*q)^*p^*$.

The other side is by

$$(p+q)^* = 1 + (p+q)(p+q)^* = (1 + q(p+q)^*) + p(p+q)^*.$$

Because of the inductive star law there is

$$p^* + p^*q(p+q)^* = p^*(1 + q(p+q)^*) \leq (p+q)^*.$$

Apply it once more, we eventually get $(p^*q)^*p^* \leq (p+q)^*$.

- $((p+q)^* = p^*(qp^*)^*)$: By sliding there is $p^*(qp^*)^* = (p^*q)^*p^* = (p+q)^*$.
- $(0 \leq p)$: Note that $0 + 1 \cdot p = p \leq p$. Apply the inductive star law, and we have $0 = 1^* \cdot 0 \leq p$.

Rules in Figure 8.2b can be derived by:

- (unrolling): For $\leq$ side, applying fixed-point twice on p^*, we have

$$1 + p + (pp)p^* = p^*.$$

  Applying the inductive star law, we have $(pp)^*(1+p) \leq p^*$.

  For $\geq$ side, applying fixed-point on $(pp)^*$ we have

$$1 + (pp)^*(1+p)p = (pp)^*p + (1 + (pp)^*pp) = (pp)^*(1+p).$$

  Then by the inductive law, we have $p^* \leq (pp)^*(1+p)$.

- (swap-star): Applying fixed-point there is

$$p^*q = q + p^*pq = q + p^*qp.$$

  By the inductive star law there is $qp^* \leq p^*q$.

  Similarly, the other side is by

$$qp^* = q + qpp^* = q + pqp^*,$$

  which leads to $p^*q \leq qp^*$.

- (star-rewrite): By fixed-point there is

$$r^*p = p + r^*rp = p + r^*pq.$$

Applying the inductive star law there is $pq^* \leq r^*p$.

To prove the other side, note that with fixed-point there is

$$pq^* = p + pqq^* = p + rpq^*.$$

The inductive star law gives $r^*p \leq pq^*$.

□

Research on formal power series dates back to [222], and see also some recent references [223, 224].

Formal power series generalize formal languages by weighing strings with the extended natural number $\overline{\mathbb{N}}$.

Definition 8.2.4. *The* extended set of natural numbers *is* $\overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$*, where* ∞ *is an added top element. The calculation in this semiring follows the correspondences in* $\mathbb{N}$*, and:*

$$0 + \infty = \infty, \quad 0 \cdot \infty = \infty \cdot 0 = 0, \quad 0^* = 1;$$

$$\forall n \in \overline{\mathbb{N}} \backslash \{0\} : \quad n + \infty = \infty, \quad n \cdot \infty = \infty \cdot n = \infty, \quad n^* = \infty.$$

A countable summation $\sum_{i \in I} n_i$ *for* $n_i \in \overline{\mathbb{N}}$ *is defined to be* ∞ *if there exists an* $i_0 \in I$ *such that* $n_{i_0} = \infty$*, or if there exists infinitely many non-zero* n_i*'s. In other cases, it degenerates to a finite summation and the definition follows naturally.*

The partial order in $\overline{\mathbb{N}}$ *extends the natural partial order in* $\mathbb{N}$ *by* $\forall n \in \overline{\mathbb{N}}, n \leq \infty$.

Definition 8.2.5 ([223, 224])**.** *For a finite alphabet* Σ*, a* formal power series $\mathbf{f}$ *over* Σ *is a function*

$\mathbf{f} : \Sigma^* \to \overline{\mathbb{N}}$*, and can be represented by* $\mathbf{f} = \sum_{w \in \Sigma^*} \mathbf{f}[w]w$ *where* $\mathbf{f}[w] \in \overline{\mathbb{N}}$ *is the* coefficient *of string* w*. We denote the set of the formal power series over* Σ *by* $\overline{\mathbb{N}}\langle\langle\Sigma^*\rangle\rangle$*.*

For example, the zero mapping in $\overline{\mathbb{N}}\langle\langle\Sigma^*\rangle\rangle$ is represented by $\mathbf{f} = 0$. The unit mapping $\mathbf{f} = 1\epsilon$ maps the empty string ϵ to 1, and the others to 0. The mapping represented by $\mathbf{f} = 1a$ for $a \in \Sigma$ maps a to 1, and the others to 0.

Definition 8.2.6. *Addition, multiplication and the star operation are defined on* $\overline{\mathbb{N}}\langle\langle\Sigma^*\rangle\rangle$ *by*

$$(\mathbf{f} + \mathbf{g})[w] = \mathbf{f}[w] + \mathbf{g}[w], \tag{8.2.0.2}$$

$$(\mathbf{f} \cdot \mathbf{g})[w] = \sum_{uv=w} \mathbf{f}[u]\mathbf{g}[v], \tag{8.2.0.3}$$

$$(\mathbf{f}^*)[w] = \sum_{n \geq 0} \sum_{u_1 \cdots u_n = w} \mathbf{f}[u_1] \cdots \mathbf{f}[u_n]w. \tag{8.2.0.4}$$

Here uv *is the concatenation of strings in* Σ^**, and* u_i *can be the empty string* ϵ *in (8.2.0.4). Note also that* $\mathbf{f}^* = \sum_{n \geq 0} \mathbf{f}^n$*.*

The partial order in $\overline{\mathbb{N}}\langle\langle\Sigma^*\rangle\rangle$ *is defined by:*

$$\mathbf{f} \leq \mathbf{g} \leftrightarrow \forall w \in \Sigma^*, \mathbf{f}[w] \leq \mathbf{g}[w]. \tag{8.2.0.5}$$

With these operations in formal power series, it is possible to interpret expressions over Σ as formal power series over Σ by a semantic mapping $\{\!\{-\}\!\}$.

Definition 8.2.7. $\{\!\{-\}\!\} : \mathrm{Exp}_\Sigma \to \overline{\mathbb{N}}\langle\langle\Sigma^*\rangle\rangle$ *is defined by*

$$\{\!\{0\}\!\} = 0, \qquad \{\!\{a\}\!\} = 1a, \qquad \{\!\{e+f\}\!\} = \{\!\{e\}\!\} + \{\!\{f\}\!\},$$

$$\{\!\{1\}\!\} = 1\epsilon, \qquad \{\!\{e^*\}\!\} = \{\!\{e\}\!\}^*, \qquad \{\!\{e \cdot f\}\!\} = \{\!\{e\}\!\} \cdot \{\!\{f\}\!\},$$

where $a \in \Sigma$*, and* $e, f \in \mathrm{Exp}_\Sigma$.

Then we are able to define rational power series as an analogue to regular languages.

Definition 8.2.8 ([223, 224])**.** *The set of* rational power series*, denoted by* $\overline{\mathbb{N}}^{\mathrm{rat}}\langle\langle\Sigma^*\rangle\rangle$*, is the smallest subset of* $\overline{\mathbb{N}}\langle\langle\Sigma^*\rangle\rangle$ *containing:* (1) $\mathbf{f} = 0$*;* (2) $\mathbf{f} = 1\epsilon$*;* (3) $\mathbf{f} = 1a$ *for all* $a \in \Sigma$*, and is closed under* $+, \cdot, *$.

A series of works from Ésik and Kuich [215], Béal et al. [225, 226], Bloom and Ésik [227] demonstrates the rational power series as a pivotal model for the NKA axioms.

Theorem 8.2.9 ([215, 227])**.** *The NKA axioms are sound and complete for* $(\overline{\mathbb{N}}^{\mathrm{rat}}\langle\langle\Sigma^*\rangle\rangle, +, \cdot, *, \leq, 0, 1\epsilon)$*. Namely, for any expression* e *and* f *over* Σ*, we have*

$$\vdash_{\mathrm{NKA}} e = f \quad \Leftrightarrow \quad \{\!\{e\}\!\} = \{\!\{f\}\!\}. \tag{8.2.0.6}$$

Remark 8.2.1 (Complexity related to NKA)**.** *Bloom and Ésik [227] have proposed an algorithm to determine the equivalence of two rational power series, so the equational theory of NKA is decidable. Meanwhile, a subset* $1^*\mathcal{K} = \{1^*p \ : \ p \in \mathcal{K}\}$ *satisfies the Kleene algebra axioms, and the equational theory of KA is* PSPACE*-complete [228], thus equational theory of NKA is also* PSPACE*-hard. However, by linking formal power series to weighted finite automata, Eilenberg [229] shows that it is undecidable whether a given inequality* $e \leq f$ *holds in NKA.*

8.3 Quantum path model

To address the infinity issue, we introduce a generalization of quantum superoperators in this section, named quantum path model, a sound model of NKA. We include detailed quantum preliminaries in Section 8.3.1, introduce extended positive operators as a generalization of quantum states in Section 8.3.2, define the quantum path model as an analog of the path integral in quantum mechanics in Section 8.3.3, and embed quantum superoperators in the quantum path model in Section 8.3.4. We recommend that first-time readers skip technical construction details in this section.

8.3.1 Quantum preliminaries

We review basic notations from quantum information that are used in this chapter. Curious readers should refer to [230, 231] for more details.

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 $|\psi\rangle$ and $|\varphi\rangle$ is denoted by $\langle\psi|\varphi\rangle$, which is the product of the Hermitian conjugate of $|\psi\rangle$, denoted by $\langle\psi|$, and the vector $|\varphi\rangle$.

Linear operators between n-dimensional Hilbert spaces are represented by $n \times n$ matrices. For example, the zero operator $O_{\mathcal{H}}$ and the identity operator $I_{\mathcal{H}}$ can be identified by the zero matrix and the identity matrix on $\mathcal{H}$. The Hermitian conjugate of operator A is denoted by $A^\dagger$. Operator A is *positive semidefinite* if for all vectors $|\psi\rangle \in \mathcal{H}$, $\langle\psi| A |\psi\rangle \geq 0$. The set of positive semidefinite operators over $\mathcal{H}$ is denoted by $\mathcal{PO}(\mathcal{H})$. This gives rise to the *Löwner order* $\sqsubseteq$ among operators: $A \sqsubseteq B \Leftrightarrow B - A$ is positive semidefinite.

A *density operator* ρ is a positive semidefinite operator $\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|$ where $\sum_i p_i = 1, p_i > 0$. A special case $\rho = |\psi\rangle\langle\psi|$ is conventionally denoted as $|\psi\rangle$. A positive semidefinite operator ρ on $\mathcal{H}$ is a *partial* density operator if $\mathrm{tr}(\rho) \leq 1$, where $\mathrm{tr}(\rho)$ is the matrix trace of ρ. The set of partial density operators is denoted by $\mathcal{D}(\mathcal{H})$.

A superoperator $\mathcal{E}$ is a mapping from $\mathcal{D}(\mathcal{H})$ to $\mathcal{D}(\mathcal{H}')$ for Hilbert spaces $\mathcal{H}, \mathcal{H}'$. It is *positive* if it maps from $\mathcal{D}(\mathcal{H})$ to $\mathcal{D}(\mathcal{H}')$ for Hilbert spaces $\mathcal{H}, \mathcal{H}'$. It is *completely-positive* if for any Hilbert space $\mathcal{A}$, the superoperator $\mathcal{E} \otimes I_{\mathcal{A}}$ is positive. It is *trace-non-increasing* if for any initial state $\rho \in \mathcal{D}(\mathcal{H})$, the final state $\mathcal{E}(\rho) \in \mathcal{D}(\mathcal{H}')$ satisfies $\mathrm{tr}(\mathcal{E}(\rho)) \leq \mathrm{tr}(\rho)$. The evolution of a quantum system can be characterized by a *completely-positive* and *trace-non-increasing* linear superoperator $\mathcal{E}$. We denote the set of such superoperators by $\mathcal{QC}(\mathcal{H}, \mathcal{H}')$. The special case, when $\mathcal{H}' = \mathcal{H}$, is denoted by $\mathcal{QC}(\mathcal{H})$.

For two superoperators $\mathcal{E}_1, \mathcal{E}_2 \in \mathcal{QC}(\mathcal{H})$, the composition is defined as $(\mathcal{E}_1 \circ \mathcal{E}_2)(\rho) = \mathcal{E}_2(\mathcal{E}_1(\rho))$. If there exists $\mathcal{E}$ and $\mathcal{E}_i \in \mathcal{QC}(\mathcal{H})$ satisfying $\mathcal{E}(\rho) = \sum_i \mathcal{E}_i(\rho)$ for every $\rho \in \mathcal{PO}(\mathcal{H})$, then we define $\mathcal{E}$ as $\sum_i \mathcal{E}_i$. For every superoperator $\mathcal{E} \in \mathcal{QC}(\mathcal{H}, \mathcal{H}')$, by [232] 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 Schrödinger-Heisenberg *dual* of a superoperator $\mathcal{E}(\rho) = \sum_k E_k \rho E_k^\dagger$ is $\mathcal{E}^\dagger(\rho) = \sum_k E_k^\dagger \rho E_k$.

A quantum *measurement* on a system over Hilbert space $\mathcal{H}$ can be described by a set of linear operators $\{M_m\}_m$ where $\sum_m M_m^\dagger M_m = I_{\mathcal{H}}$. The measurement outcome m is observed with probability $p_m = \mathrm{tr}\big(M_m \rho M_m^\dagger\big)$ for each m, which will collapse the pre-measure state ρ to $\mathcal{M}_m(\rho) = M_m \rho M_m^\dagger / p_m$. A quantum measurement is *projective* if $M_i M_j = M_i$ if $i = j$ and $O_{\mathcal{H}}$ otherwise. Namely, all M_i are projective operators orthogonal to each other.

8.3.2 Extended positive operators

The set $\mathcal{PO}(\mathcal{H})$ does not contain any infinity. We need to incorporate different infinities into it to distinguish different path sets which may lead to different divergent summations.

Definition 8.3.1. *A series of $\mathcal{PO}(\mathcal{H})$ is a countable multiset of $\mathcal{PO}(\mathcal{H})$, and can be written as $\biguplus_{i\in I}\rho_i$, where I is a countable index set. Symbol $\biguplus_{i\in I}$ enumerates every element ρ_i in the multiset. The set of series of $\mathcal{PO}(\mathcal{H})$ is denoted by $\mathcal{S}(\mathcal{H})$.*

The union *of countably many series is denoted by:*

$$\biguplus_{i\in I}\left(\biguplus_{j\in J_i}\rho_{ij}\right)=\biguplus_{(i,j):i\in I,j\in J_i}\rho_{ij}. \tag{8.3.2.1}$$

Note $\biguplus_{i\in I}\biguplus_{j\in J_i}\rho_{ij}\in\mathcal{S}(\mathcal{H})$ since the index set is countable.

A binary relation $\lesssim$ *over $\mathcal{S}(\mathcal{H})$ is defined by: $\biguplus_{i\in I}\rho_i\lesssim\biguplus_{j\in J}\sigma_j$ if and only if for every $\epsilon>0$ and finite $I'\subseteq I$, there exists a finite $J'\subseteq J$, such that*

$$\sum\nolimits_{i\in I'}\rho_i\sqsubseteq\epsilon I_{\mathcal{H}}+\sum\nolimits_{j\in J'}\sigma_j. \tag{8.3.2.2}$$

We induce another binary relation $\sim$ from $\lesssim$ on $\mathcal{S}(\mathcal{H})$ by:

$$\biguplus_{i\in I}\rho_i\sim\biguplus_{j\in J}\sigma_j\quad\Leftrightarrow\quad\biguplus_{i\in I}\rho_i\lesssim\biguplus_{j\in J}\sigma_j\ \wedge\ \biguplus_{j\in J}\sigma_j\lesssim\biguplus_{i\in I}\rho_i.$$

Symbol $\biguplus_{i\in I}$ is employed to distinguish the series from the normal summation $\sum_{i\in I}$ over $\mathcal{PO}(\mathcal{H})$. We will build connections between these two notions so that $\biguplus_{i\in I}$ can readily help us in the analysis of convergence, and more.

We represent a finite series by enumerating its elements. Like a series with one element $O_{\mathcal{H}}$, we denote it by $\{\!|O_{\mathcal{H}}|\!\}$.

The definition of $\lesssim$ aims at a generalization to the Löwner order in $\mathcal{S}(\mathcal{H})$ that distinguishes the different infinities while preserving relations like $\{\!|I_{\mathcal{H}}|\!\} \lesssim \biguplus_{i>0} \frac{1}{2^i} I_{\mathcal{H}}$, whose correspondence in $\mathcal{PO}(\mathcal{H})$ holds.

Lemma 8.3.2. *$\lesssim$ is a preorder, so $\sim$ is an equivalence relation.*

Proof of Lemma 8.3.2. Reflexivity is proved by choosing $J' = I'$ in the definition. To prove transitivity, we assume $\biguplus_{i\in I} \rho_i \lesssim \biguplus_{j\in J} \sigma_j$ and $\biguplus_{j\in J} \sigma_j \lesssim \biguplus_{k\in K} \gamma_k$. For $\epsilon > 0$ and finite $I' \subseteq I$, there exists a finite $J' \subseteq J$ such that $\sum_{i\in I'} \rho_i \sqsubseteq \frac{\epsilon}{2} I_{\mathcal{H}} + \sum_{j\in J'} \sigma_j$. Then there exists a finite $K' \subseteq K$ such that $\sum_{j\in J'} \sigma_j \sqsubseteq \frac{\epsilon}{2} I_{\mathcal{H}} + \sum_{k\in K'} \gamma_k$ as well. Because $\sqsubseteq$ is monotone with respect to $+$, we have $\sum_{i\in I'} \rho_i \sqsubseteq \epsilon I_{\mathcal{H}} + \sum_{k\in K'} \gamma_k$. $\square$

Several facts about $\sim$ are deposited here.

Lemma 8.3.3. *We demonstrate several basic facts about $\mathcal{S}(\mathcal{H})$.*

(i) If for all $i \in I$, $\biguplus_{j\in J_i} \rho_{ij} \lesssim \biguplus_{k\in K_i} \sigma_{ik}$, then

$$\biguplus_{i\in I} \biguplus_{j\in J_i} \rho_{ij} \lesssim \biguplus_{i\in I} \biguplus_{k\in K_i} \sigma_{ik}. \tag{8.3.2.3}$$

(ii) Let $n_i \in \overline{\mathbb{N}}$ for all $i \in I$. Then for all $\biguplus_{j\in J} \rho_j \in \mathcal{S}(\mathcal{H})$, there is

$$\biguplus_{i\in I} \biguplus_{0\le k<n_i} \biguplus_{j\in J_i} \rho_j \sim \biguplus_{0\le k<\sum_{i\in I} n_i} \biguplus_{j\in J_i} \rho_j. \tag{8.3.2.4}$$

Here $\{k : 0 \le k < \infty\} = \mathbb{N}$.

(iii) If $\sum_{i\in I}\rho_i$ converges in $\mathcal{PO}(\mathcal{H})$, then

$$\biguplus_{i\in I}\rho_i \sim \left\{\left|\sum_{i\in I}\rho_i\right|\right\}. \tag{8.3.2.5}$$

(iv) For a series $\biguplus_{i\in\mathbb{N}}\biguplus_{j\in J_i}\rho_{ij}\in\mathcal{S}(\mathcal{H})$, if there exists $\biguplus_{k\in K}\sigma_k$ such that for all $n\geq 0$,

$$\biguplus_{0\leq i<n}\biguplus_{j\in J_i}\rho_{ij}\lesssim\biguplus_{k\in K}\sigma_k, \tag{8.3.2.6}$$

then $\biguplus_{i\in\mathbb{N}}\biguplus_{j\in J_i}\rho_{ij}\lesssim\biguplus_{k\in K}\sigma_k$.

(v) If $\biguplus_{i\in I}\rho_i\lesssim\biguplus_{j\in J}\sigma_j$, then for $\mathcal{E}\in\mathcal{QC}(\mathcal{H})$, there is

$$\biguplus_{i\in I}\mathcal{E}(\rho_i)\lesssim\biguplus_{j\in J}\mathcal{E}(\sigma_j). \tag{8.3.2.7}$$

Proof of Lemma 8.3.3. W.l.o.g. we assume the index sets to be subsets of $\mathbb{N}$.

(i) For any $\epsilon>0$ and any finite subseries $\biguplus_{i\in I,j\in J'_i}\rho_{ij}$ of $\biguplus_{i\in I}\biguplus_{j\in J_i}\rho_{ij}$, there exists an N such that for $i\geq N$, there is $J'_i=\phi$. When $N=0$, then $\{(i,j):i\in I,j\in J'_i\}=\phi$ and the inequality holds with an empty subset chosen on the right hand side. Otherwise let $\epsilon'=\frac{\epsilon}{N}$, so there exist finite index set K'_i for each $0\leq i<N$ such that $\sum_{j\in J'_i}\rho_{ij}\sqsubseteq\epsilon' I+\sum_{k\in K'_i}\sigma_{ik}$. Adding them up gives $\sum_{0\leq i<N,j\in J'_i}\rho_{ij}\sqsubseteq\epsilon I+\sum_{0\leq i<N,k\in K'_i}\sigma_{ik}$. This concludes $\biguplus_i\biguplus_{j\in J_i}\rho_{ij}\lesssim\biguplus_i\biguplus_{k\in K_i}\sigma_{ik}$.

(ii) By reordering the multisets it holds apparently.

(iii) ($\lesssim$): Notice that for any finite $I'\subseteq I$, $\sum_{i\in I'}\rho_i\sqsubseteq\sum_{i\in I}\rho_i$. Then this direction comes from

the definition.

($\gtrsim$): Since $\sum_{i\in I}\rho_i$ converges, for any $\epsilon > 0$ there is an $N > 0$ such that $\left\|\sum_{i\in I, i>N}\rho_i\right\| \leq \epsilon$, where $\|\cdot\|$ is the spectral norm. Hence $\sum_{i\in I}\rho_i \sqsubseteq \epsilon I_{\mathcal{H}} + \sum_{i\in I, i\leq N}\rho_i$. This gives $\gtrsim$ direction.

(iv) Consider any finite subseries $\biguplus_{i\in\mathbb{N}, j\in J_i'}\rho_{ij}$ selected from $\biguplus_{i\geq 0}\biguplus_{j\in J_i}\rho_{ij}$. There exists N such that for all $i \geq N$, $J_i' = \phi$. Let $n = N$ in the assumption, then we know that for any $\epsilon > 0$ there exists a finite $K' \subseteq K$ such that $\sum_{0\leq i<N, j\in J_i'}\rho_{ij} \sqsubseteq \epsilon I_{\mathcal{H}} + \sum_{k\in K'}\sigma_k$, and this concludes the proof.

(v) If $\mathcal{E}(I_{\mathcal{H}}) = O_{\mathcal{H}}$, then $\mathcal{E} \equiv O_{\mathcal{H}}$, and we are done by definition. Now we assume $\mathcal{E}(I_{\mathcal{H}}) \neq O_{\mathcal{H}}$. For every finite $I' \subseteq I$ and $\epsilon > 0$, there exists $J' \subseteq J$ such that $\sum_{i\in I'}\rho_i \sqsubseteq \frac{\epsilon}{\|\mathcal{E}(I_{\mathcal{H}})\|}I_{\mathcal{H}} + \sum_{j\in J'}\sigma_j$. Then

$$\begin{aligned}\sum_{i\in I'}\mathcal{E}(\rho_i) = \mathcal{E}\left(\sum_{i\in I'}\rho_i\right) &\sqsubseteq \mathcal{E}\left(\frac{\epsilon}{\|\mathcal{E}(I_{\mathcal{H}})\|}I_{\mathcal{H}} + \sum_{j\in J'}\sigma_j\right)\\ &\sqsubseteq \epsilon I_{\mathcal{H}} + \sum_{j\in J'}\mathcal{E}(\sigma_j).\end{aligned}$$

Here $\|\cdot\|$ is the spectral norm. This leads to $\biguplus_{i\in I}\mathcal{E}(\rho_i) \lesssim \biguplus_{j\in J}\mathcal{E}(\sigma_j)$.

□

Definition 8.3.4. *We define the extended positive operators $\mathcal{PO}_\infty(\mathcal{H}) = \mathcal{S}(\mathcal{H})/\sim$ as the set of equivalence classes of $\sim$. Let the equivalence class including $\biguplus_{i\in I}\rho_i$ be*

$$\left[\biguplus_{i\in I}\rho_i\right] = \left\{\biguplus_{j\in J}\sigma_j \,\middle|\, \biguplus_{j\in J}\sigma_j \sim \biguplus_{i\in I}\rho_i\right\}, \tag{8.3.2.8}$$

where on the right hand side is a set of series.

A partial order $\leq$ over $\mathcal{PO}_\infty(\mathcal{H})$ is induced from the preorder $\lesssim$ over $\mathcal{S}(\mathcal{H})$ by:

$$\left[\biguplus_{i\in I}\rho_i\right] \leq \left[\biguplus_{j\in J}\sigma_j\right] \;\Leftrightarrow\; \biguplus_{i\in I}\rho_i \lesssim \biguplus_{j\in J}\sigma_j. \tag{8.3.2.9}$$

We define countable summation over $\mathcal{PO}_\infty(\mathcal{H})$ from the union in $\mathcal{S}(\mathcal{H})$ by

$$\sum\nolimits_{i\in I}\left[\biguplus_{j\in J_i}\rho_{ij}\right] = \left[\biguplus_{i\in I}\biguplus_{j\in J_i}\rho_{ij}\right]. \tag{8.3.2.10}$$

The summation defined above is independent of the choices of $\biguplus_{j\in J_i}\rho_{ij}$ because of Lemma 8.3.3.(i).

We slightly abuse notation, writing $[\rho]$ to represent $[\{\!|\rho|\!\}]$ for $\rho \in \mathcal{PO}(\mathcal{H})$. A frequently used case of (8.3.2.10) is to write the equivalence class of a series as

$$\left[\biguplus_{i\in I}\rho_i\right] = \sum\nolimits_{i\in I}[\rho_i], \tag{8.3.2.11}$$

where we can intuitively deem the countable summation over $\mathcal{PO}_\infty(\mathcal{H})$ as a generalized summation over $\mathcal{PO}(\mathcal{H})$. For example, we have $\sum_{i>0}\left[\frac{1}{2^i}I_{\mathcal{H}}\right] = \left[\sum_{i>0}\frac{1}{2^i}I_{\mathcal{H}}\right] = [I_{\mathcal{H}}]$ according to Lemma 8.3.3.(iii).

Remark 8.3.1. *$\mathcal{PO}(\mathcal{H})$ is embedded in $\mathcal{PO}_\infty(\mathcal{H})$ by $\rho \mapsto [\rho]$ as finite positive operators. Besides these, $\mathcal{PO}_\infty(\mathcal{H})$ contains distinguishable divergent summations unattainable by $\mathcal{PO}(\mathcal{H})$: e.g., $\sum_{i>0}[|0\rangle\langle 0|]$ is different from $\sum_{i>0}[|1\rangle\langle 1|]$, and less than $\sum_{i>0}[I_{\mathcal{H}_2}]$. These divergent summations are leveraged to depict the domain and the range of our extended quantum superoperators.*

8.3.3 Quantum actions

We are now ready to introduce quantum actions, a generalization of superoperators in the quantum path model, inspired by the path integral formulation of quantum mechanics.

Definition 8.3.5. *A* quantum action*, or* action *for simplicity, over $\mathcal{PO}_\infty(\mathcal{H})$ is a mapping from $\mathcal{PO}_\infty(\mathcal{H})$ to $\mathcal{PO}_\infty(\mathcal{H})$.*

A quantum action $\mathcal{A}$ is linear *if for series $\sum_{j\in J_i}[\rho_{ij}]$,*

$$\mathcal{A}\left(\sum_{i\in I}\sum_{j\in J_i}[\rho_{ij}]\right)=\sum_{i\in I}\mathcal{A}\left(\sum_{j\in J_i}[\rho_{ij}]\right). \tag{8.3.3.1}$$

A quantum action $\mathcal{A}$ is monotone *if for any two series $\sum_{i\in I}[\rho_i]\leq\sum_{j\in J}[\sigma_j]$,*

$$\mathcal{A}\left(\sum_{i\in I}[\rho_i]\right)\leq\mathcal{A}\left(\sum_{j\in J}[\sigma_j]\right). \tag{8.3.3.2}$$

We denote the set of linear and monotone quantum actions over $\mathcal{PO}_\infty(\mathcal{H})$ by $\mathcal{P}(\mathcal{H})$ as the set of quantum path actions.

The zero action $\mathcal{O}_\mathcal{H}$ maps everything to $[O_\mathcal{H}]$, and the identity action is denoted by $\mathcal{I}_\mathcal{H}$.

A physical interpretation of quantum path actions in $\mathcal{P}(\mathcal{H})$ is the collection of quantum evolution along a single or many possible trajectories of the underlying system. Thus, one can readily define the composition and the sum of quantum path actions, as the concatenation and the union of trajectories.

Definition 8.3.6. *We define the operations in $\mathcal{P}(\mathcal{H})$ by:*

$$\left(\sum_{i\in I}\mathcal{A}_i\right)\left(\sum_{j\in J}[\rho_j]\right)=\sum_{i\in I}\mathcal{A}_i\left(\sum_{j\in J}[\rho_j]\right), \tag{8.3.3.3}$$

$$(\mathcal{A}_1;\mathcal{A}_2)\left(\sum\nolimits_{j\in J}[\rho_j]\right)=\mathcal{A}_2\left(\mathcal{A}_1\left(\sum\nolimits_{j\in J}[\rho_j]\right)\right), \tag{8.3.3.4}$$

$$\mathcal{A}^*=\sum\nolimits_{i\geq 0}\mathcal{A}^i. \tag{8.3.3.5}$$

Here $\mathcal{A}^i=\mathcal{I}_{\mathcal{H}};\mathcal{A};\mathcal{A};\cdots;\mathcal{A}$ where $\mathcal{A}$ repeats i times.

Additionally, we define $\mathcal{A}_1\diamond\mathcal{A}_2=\mathcal{A}_2;\mathcal{A}_1$.

A point-wise partial order $\preceq$ in $\mathcal{P}(\mathcal{H})$ is induced point-wisely: $\mathcal{A}_1\preceq\mathcal{A}_2$ if and only if

$$\forall\sum\nolimits_{i\in I}[\rho_i],\ \mathcal{A}_1\left(\sum\nolimits_{i\in I}[\rho_i]\right)\leq\mathcal{A}_2\left(\sum\nolimits_{i\in I}[\rho_i]\right). \tag{8.3.3.6}$$

Lemma 8.3.7. *$\sum_i$, ; and $*$ operations are closed in $\mathcal{P}(\mathcal{H})$.*

Proof of Lemma 8.3.7. The monotone of $\sum_i$ follows Lemma 8.3.3.(i), and the monotone of ; follows the definition. It suffices to verify the linearity of them.

For $\sum_i$, notice that

$$\begin{aligned}\left(\sum_k\mathcal{A}_k\right)\left(\sum_i\sum_{j\in J_i}[\rho_{ij}]\right)&=\sum_k\mathcal{A}_k\left(\sum_i\sum_{j\in J_i}[\rho_{ij}]\right)\\&=\sum_k\sum_i\mathcal{A}_k\left(\sum_{j\in J_i}[\rho_{ij}]\right)\\&=\sum_i\sum_k\mathcal{A}_k\left(\sum_{j\in J_i}[\rho_{ij}]\right)\\&=\sum_i\left(\sum_k\mathcal{A}_k\right)\left(\sum_{j\in J_i}[\rho_{ij}]\right).\end{aligned}$$

For ; operation, it is directly proved by

$$
\begin{aligned}
(\mathcal{A}_1;\mathcal{A}_2)\left(\sum_i\sum_{j\in J_i}[\rho_{ij}]\right) &= \mathcal{A}_2\left(\sum_i \mathcal{A}_1\left(\sum_{j\in J_i}[\rho_{ij}]\right)\right)\\
&= \sum_i \mathcal{A}_2\left(\mathcal{A}_1\left(\sum_{j\in J_i}[\rho_{ij}]\right)\right)\\
&= \sum_i (\mathcal{A}_1;\mathcal{A}_2)\left(\sum_{j\in J_i}[\rho_{ij}]\right).
\end{aligned}
$$

□

Our main result is that $\mathcal{P}(\mathcal{H})$ with the above partial order and operations satisfies the axioms of NKA. Since infinite summations are well-defined over quantum path actions, any NKA derivation safely induces a derivation over quantum path actions without worrying about the infinity issue.

Theorem 8.3.8. *The NKA axioms are sound for the* quantum path model*, defined by* $(\mathcal{P}(\mathcal{H}), +, ;, *, \preceq, \mathcal{O}_{\mathcal{H}}, \mathcal{I}_{\mathcal{H}})$*. Here* $+$ *is the* $\sum_i$ *operation restricted on two operands.*

Proof of Theorem 8.3.8. The proofs of monotone of $+$ and ; operations, the star laws are presented here.

- $p \leq q \wedge r \leq s \rightarrow p + r \leq q + s$: First we show that $+$ and $\leq$ over $\mathcal{PO}_\infty(\mathcal{H})$ follow this rule.

  Let $\uplus$ be an abbreviation of $\biguplus_i$ where there are only two operands.

  For $\sum_{i\in I}[\rho_i] \leq \sum_{j\in J}[\sigma_j]$ and $\sum_{k\in K}[\gamma_k] \leq \sum_{l\in L}[\chi_l]$, notice that $\biguplus_{i\in I}\rho_i \lesssim \biguplus_{j\in J}\sigma_j$ and $\biguplus_{k\in K}\gamma_k \lesssim \biguplus_{l\in L}\chi_l$. By Lemma 8.3.3.(i) there is $\biguplus_{i\in I}\rho_i \uplus \biguplus_{k\in K}\gamma_k \lesssim \biguplus_{j\in J}\sigma_j \uplus \biguplus_{l\in L}\chi_l$.

Hence

$$\sum_{i\in I}[\rho_i]+\sum_{k\in K}[\gamma_k]=\left[\biguplus_{i\in I}\rho_i\uplus\biguplus_{k\in K}\gamma_k\right]$$
$$\leq\left[\biguplus_{j\in J}\sigma_j\uplus\biguplus_{l\in L}\chi_l\right]=\sum_{j\in J}[\sigma_j]+\sum_{l\in L}[\chi_l]\,.$$

Then at $\mathcal{P}(\mathcal{H})$ level, the inequality holds by definition.

- $p\leq q\wedge r\leq s\rightarrow pr\leq qs$: Because $\mathcal{A}\in\mathcal{P}(\mathcal{H})$ is monotone, by definition this law holds.

- $1+pp^*\leq p^*$: For any $\mathcal{A}\in\mathcal{P}(\mathcal{H})$, there is

$$\begin{aligned}\mathcal{I}_{\mathcal{H}}+(\mathcal{A};\mathcal{A}^*)&=\mathcal{A}^0+\left(\mathcal{A};\sum_{i\geq 0}\mathcal{A}^i\right)\\&=\mathcal{A}^0+\sum_{i\geq 0}(\mathcal{A};\mathcal{A}^i)\\&=\sum_{i\geq 0}\mathcal{A}^i=\mathcal{A}^*.\end{aligned}$$

The second equality comes from the definition of ; operation.

- $*$-continuity: Lemma 8.3.3.(iv) leads to the $*$-continuity in $\mathcal{PO}_\infty(\mathcal{H})$: for $\sum_{i\in\mathbb{N}}\sum_{j\in J_i}[\rho_{ij}]$, if there exists $\sum_{k\in K}[\sigma_k]$ such that for all $n\geq 0:\sum_{0\leq i<n}\sum_{j\in J_i}[\rho_{ij}]\leq\sum_{k\in K}[\sigma_k]$, then $\sum_{i\in\mathbb{N}}\sum_{j\in J_i}[\rho_{ij}]\leq\sum_{k\in K}[\sigma_k]$.

Eventually we show the $*$-continuity of $\mathcal{P}(\mathcal{H})$. For $\mathcal{A}_p,\mathcal{A}_q,\mathcal{A}_r,\mathcal{A}_s$ satisfying $\sum_{0\leq i<n}(\mathcal{A}_p;\mathcal{A}_q^i;\mathcal{A}_r)\preceq\mathcal{A}_s$ for all $n\geq 0$, there is $\sum_{0\leq i<n}(\mathcal{A}_p;\mathcal{A}_q^i;\mathcal{A}_r)(\sum_{j\in J}[\rho_j])\leq\mathcal{A}_s(\sum_{j\in J}[\rho_j])$ for every

$\sum_{j\in J}[\rho_j] \in \mathcal{PO}_\infty(\mathcal{H})$. By the $*$-continuity in $\mathcal{PO}_\infty(\mathcal{H})$ and linearity, inequality

$$\begin{aligned}&(\mathcal{A}_p;\mathcal{A}_q^*;\mathcal{A}_r)\left(\sum_{j\in J}[\rho_j]\right)\\ =&\sum_{i\geq 0}(\mathcal{A}_p;\mathcal{A}_q^i;\mathcal{A}_r)\left(\sum_{j\in J}[\rho_j]\right)\leq \mathcal{A}_s\left(\sum_{j\in J}[\rho_j]\right)\end{aligned}$$

holds for every $\sum_{j\in J}[\rho_j] \in \mathcal{PO}_\infty(\mathcal{H})$. This concludes the $*$-continuity rule in $\mathcal{P}(\mathcal{H})$. Then the inductive star laws follow naturally.

□

8.3.4 Embedding of $\mathcal{QC}(\mathcal{H})$ in $\mathcal{P}(\mathcal{H})$

We mentioned the intuition that quantum path actions are generalizations of quantum superoperators in the quantum path model. We now make it precise by building an embedding from quantum superoperators to quantum path actions (and hence the quantum path model), which allows us to prove superoperator equations via NKA theorems.

Definition 8.3.9. Path lifting *is a mapping from* $\mathcal{E} \in \mathcal{QC}(\mathcal{H})$ *to a quantum path action* $\langle\mathcal{E}\rangle^\uparrow$: $\sum_{i\in I}[\rho_i] \mapsto \sum_{i\in I}[\mathcal{E}(\rho_i)]$.

$\langle\mathcal{E}\rangle^\uparrow$ is well-defined (it does not depend on the choices of $\sum_{i\in I}[\rho_i]$) because of Lemma 8.3.3.(v).

The path lifting embeds $\mathcal{QC}(\mathcal{H})$ in $\mathcal{P}(\mathcal{H})$ by the following lemma, whose proof is routine.

Lemma 8.3.10. *The path lifting has the following properties:*

(i) $\langle\mathcal{E}\rangle^\uparrow \in \mathcal{P}(\mathcal{H})$, *for* $\mathcal{E} \in \mathcal{QC}(\mathcal{H})$.

(ii) $\mathcal{E}_1 = \mathcal{E}_2 \Leftrightarrow \langle\mathcal{E}_1\rangle^\uparrow = \langle\mathcal{E}_2\rangle^\uparrow$, *for* $\mathcal{E}_1, \mathcal{E}_2 \in \mathcal{QC}(\mathcal{H})$.

(iii) operations $\circ$ and $\sum_i$ (when defined) in $\mathcal{QC}(\mathcal{H})$ are preserved by path lifting as ; and $\sum_i$ operations in $\mathcal{P}(\mathcal{H})$.

Proof of Lemma 8.3.10.

(i) By Lemma 8.3.3.(v), $\langle\mathcal{E}\rangle^{\uparrow}$ is monotone. Linearity is from

$$\langle\mathcal{E}\rangle^{\uparrow}\left(\sum_i\sum_{j\in J_i}[\rho_{ij}]\right)=\sum_i\sum_{j\in J_i}[\mathcal{E}(\rho_{ij})]=\sum_i\langle\mathcal{E}\rangle^{\uparrow}\left(\sum_{j\in J_i}[\rho_{ij}]\right).$$

(ii) ($\Rightarrow$): By definition this direction holds.

($\Leftarrow$): To prove the injectivity of path lifting, we assume $\mathcal{E}_1\neq\mathcal{E}_2$ while $\langle\mathcal{E}_1\rangle^{\uparrow}=\langle\mathcal{E}_2\rangle^{\uparrow}$, then there exists $\rho\in\mathcal{PO}(\mathcal{H})$ such that $\mathcal{E}_1(\rho)\neq\mathcal{E}_2(\rho)$. $\langle\mathcal{E}_1\rangle^{\uparrow}=\langle\mathcal{E}_2\rangle^{\uparrow}$ indicates that

$$[\mathcal{E}_1(\rho)]=\langle\mathcal{E}_1\rangle^{\uparrow}([\rho])=\langle\mathcal{E}_2\rangle^{\uparrow}([\rho])=[\mathcal{E}_2(\rho)].$$

Hence $\{\mathcal{E}_1(\rho)\}\sim\{\mathcal{E}_2(\rho)\}$. If $\mathcal{E}_1(\rho)=O_{\mathcal{H}}$, then for every $\epsilon>0$, there is $\mathcal{E}_2(\rho)\sqsubseteq\epsilon I_{\mathcal{H}}$, resulting in $\mathcal{E}_2(\rho)=O_{\mathcal{H}}=\mathcal{E}_1(\rho)$, which is a contradiction. If $\mathcal{E}_1(\rho)\neq O_{\mathcal{H}}$, for every $0<\epsilon<\|\mathcal{E}_1(\rho)\|$, there is $\mathcal{E}_1(\rho)\sqsubseteq\epsilon I_{\mathcal{H}}+\mathcal{E}_2(\rho)$. Hence $\mathcal{E}_1(\rho)\sqsubseteq\mathcal{E}_2(\rho)$. Similarly we have $\mathcal{E}_2(\rho)\sqsubseteq\mathcal{E}_1(\rho)$. So $\mathcal{E}_1(\rho)=\mathcal{E}_2(\rho)$ is the contradiction.

(iii) For $\mathcal{E}_1,\mathcal{E}_2\in\mathcal{QC}(\mathcal{H})$ and $\sum_{i\in I}[\rho_i]\in\mathcal{PO}_\infty(\mathcal{H})$, there is

$$(\langle\mathcal{E}_1\rangle^{\uparrow};\langle\mathcal{E}_2\rangle^{\uparrow})\left(\sum_{i\in I}[\rho_i]\right)=\langle\mathcal{E}_2\rangle^{\uparrow}\left(\sum_{i\in I}[\mathcal{E}_1(\rho_i)]\right)=\sum_{i\in I}[\mathcal{E}_2(\mathcal{E}_1(\rho_i))]=\langle\mathcal{E}_1\circ\mathcal{E}_2\rangle^{\uparrow}\left(\sum_{i\in I}[\rho_i]\right).$$

Similarly, if $\sum_i\mathcal{E}_i$ is defined in $\mathcal{QC}(\mathcal{H})$, then $\sum_i\mathcal{E}_i(\rho)$ converges for any $\rho\in\mathcal{PO}(\mathcal{H})$. By

Lemma 8.3.3.(iii), for every $\sum_{j\in J}[\rho_j] \in \mathcal{PO}_\infty(\mathcal{H})$, there is

$$\begin{aligned}\left(\sum_i \langle \mathcal{E}_i\rangle^\uparrow\right)\left(\sum_{j\in J}[\rho_j]\right) &= \sum_i \langle \mathcal{E}_i\rangle^\uparrow\left(\sum_{j\in J}[\rho_j]\right) = \sum_j\sum_i [\mathcal{E}_i(\rho_j)] \\ &= \sum_j\left[\sum_i \mathcal{E}_i(\rho_j)\right] = \langle\sum_i \mathcal{E}_i\rangle^\uparrow\left(\sum_{j\in J}[\rho_j]\right).\end{aligned}$$

□

8.4 Quantum interpretation and quantum while programs

In this section, we link expressions, quantum path actions and quantum programs by quantum interpretation (Section 8.4.1) and encoding (Section 8.4.2).

8.4.1 Quantum interpretation

We endow equations in NKA with quantum interpretations.

Definition 8.4.1. *A* quantum interpretation setting *over an alphabet* Σ *is a pair* $\mathrm{int} = (\mathcal{H}, \mathrm{eval})$ *where*

1. $\mathcal{H}$ *is a finite dimensional Hilbert space.*

2. $\mathrm{eval} : \Sigma \to \mathcal{QC}(\mathcal{H})$ *is a function to interpret symbols.*

The quantum interpretation $\mathcal{Q}_{\mathrm{int}}$ *w.r.t. a quantum interpretation setting* int *is a mapping*

from Exp_Σ *to* $\mathcal{P}(\mathcal{H})$ *where*

$$\mathcal{Q}_{\mathrm{int}}(0) = \mathcal{O}_{\mathcal{H}}, \qquad \mathcal{Q}_{\mathrm{int}}(e+f) = \mathcal{Q}_{\mathrm{int}}(e) + \mathcal{Q}_{\mathrm{int}}(f),$$

$$\mathcal{Q}_{\mathrm{int}}(1) = \mathcal{I}_{\mathcal{H}}, \qquad \mathcal{Q}_{\mathrm{int}}(e \cdot f) = \mathcal{Q}_{\mathrm{int}}(e); \mathcal{Q}_{\mathrm{int}}(f),$$

$$\mathcal{Q}_{\mathrm{int}}(a) = \langle \mathrm{eval}(a) \rangle^{\uparrow}, \qquad \mathcal{Q}_{\mathrm{int}}(e^*) = \mathcal{Q}_{\mathrm{int}}(e)^*.$$

Here $a \in \Sigma$, *and* $\langle \mathrm{eval}(a) \rangle^{\uparrow}$ *is the path lifting of* $\mathrm{eval}(a)$.

Theorem 8.4.2. *The axioms of NKA are sound and complete w.r.t. the quantum interpretation. That is, for any* $e, f \in \mathrm{Exp}_\Sigma$,

$$\vdash_{\mathrm{NKA}} e = f \quad \Leftrightarrow \quad \forall \mathrm{int}, \mathcal{Q}_{\mathrm{int}}(e) = \mathcal{Q}_{\mathrm{int}}(f). \tag{8.4.1.1}$$

The soundness comes directly from Theorem 8.3.8. The completeness proof makes use of formal power series.

Proof of Theorem 8.4.2. ($\Rightarrow$): Formally we prove it by induction on the derivation of $\vdash_{\mathrm{NKA}} e = f$. Practically it suffices to prove the soundness of the NKA axioms on the quantum path model, which is proved in Theorem 8.3.8.

($\Leftarrow$): We will establish $\vdash_{\mathrm{NKA}} e = f$ by first showing $\{\!\{e\}\!\} = \{\!\{f\}\!\}$ and then applying Theorem 8.2.9. To that end, let us consider the case of any fixed $n \in \mathbb{N}$, and show that for string w with length less than n, there is $\{\!\{e\}\!\}[w] = \{\!\{f\}\!\}[w]$.

Let $S = \{s \in \Sigma^* : |s| \leq n\}$. Because Σ and n are finite, S is a finite set. We set $\mathcal{H} = \mathrm{span}\{|s\rangle : s \in S\}$ which is finite dimensional, and $\mathrm{eval}(a)(\rho) = \sum_{s \in S} K_{a,s} \rho K_{a,s}^\dagger$, where $K_{a,s} = \frac{1}{\sqrt{\#_a}} |sa\rangle \langle s|$ for $sa \in S$, $K_{a,s} = O_{\mathcal{H}}$ for $sa \notin S$. Here $\#_a = |\{s : sa \in S\}|$ is a

normalization factor to make sure $\mathrm{eval}(a) \in \mathcal{QC}(\mathcal{H})$. For $s = a_1 a_2 \cdots a_l$, we set $\#_s = \prod_{i=1}^{l} \#_{a_i}$.

Let $\mathrm{int} = (\mathcal{H}, \mathrm{eval})$. We claim for $s \in S$ and $r \in \mathbb{R}$, there is

$$\mathcal{Q}_{\mathrm{int}}(e)([r \cdot |s\rangle \langle s|]) = \sum_{st \in S} \sum_{k=1}^{\{\!\{e\}\!\}[t]} [r/\#_t \cdot |st\rangle \langle st|] \,. \tag{8.4.1.2}$$

The proof is based on the induction on expression e, and its proof is left to the last.

Then we consider two expressions e, f such that $\mathcal{Q}_{\mathrm{int}}(e) = \mathcal{Q}_{\mathrm{int}}(f)$. We apply this action on ϵ and $r = 1$, resulting in

$$\sum_{s \in S} \sum_{k=1}^{\{\!\{e\}\!\}[s]} [1/\#_s \cdot |s\rangle \langle s|] = \sum_{s \in S} \sum_{k=1}^{\{\!\{f\}\!\}[s]} [1/\#_s \cdot |s\rangle \langle s|] \,.$$

If there exists $t \in S : \{\!\{e\}\!\}[t] < \{\!\{f\}\!\}[t]$, then there exists $m \in \mathbb{N}$ such that $\{\!\{e\}\!\}[t] < m \leq \{\!\{f\}\!\}[t]$. By selecting $I' = \{(t, k) : 0 \leq k < m\}$ in the definition of $\biguplus_{s \in S} \biguplus_{k=1}^{\{\!\{f\}\!\}[s]} 1/\#_s \cdot |s\rangle \langle s| \lesssim \biguplus_{s \in S} \biguplus_{k=1}^{\{\!\{e\}\!\}[s]} 1/\#_s \cdot |s\rangle \langle s|$, it is impossible to find a J' to satisfy definition inequality (8.3.2.2), because there are at most $\{\!\{e\}\!\}[t]$ operators that are non-zero in basis $|t\rangle \langle t|$. The cases where $\{\!\{e\}\!\}[s] > \{\!\{f\}\!\}[s]$ can be ruled out similarly. Then $\forall s \in S, \{\!\{e\}\!\}[s] = \{\!\{f\}\!\}[s]$.

Notice that the above argument holds for any $n \in \mathbb{N}$. Hence $\{\!\{e\}\!\} = \{\!\{f\}\!\}$. By Theorem 8.2.9, $\vdash_{\mathrm{NKA}} e = f$.

Now we come back to (8.4.1.2). Let us prove it by induction on e. For the base cases,

notice that

$$\mathcal{Q}_{\text{int}}(0) = \mathcal{O}_{\mathcal{H}}, \qquad \mathcal{Q}_{\text{int}}(1) = \mathcal{I}_{\mathcal{H}},$$

$$\mathcal{Q}_{\text{int}}(a)([r \cdot |s\rangle \langle s|]) = \begin{cases} [r/\#_a \cdot |sa\rangle \langle sa|], & sa \in S, \\ [O_{\mathcal{H}}], & sa \notin S. \end{cases}$$

Combined with $\{\!\{0\}\!\} = 0$, $\{\!\{1\}\!\} = 1\epsilon$ and $\{\!\{a\}\!\} = 1a$, the equation holds for the base cases.

Consider the case $e + f$. For any $s \in S$ and $r \in \mathbb{R}$, by inductive hypotheses and Lemma 8.3.3.(ii),

$$\begin{aligned}
\mathcal{Q}_{\text{int}}(e+f)([r \cdot |s\rangle \langle s|]) &= \mathcal{Q}_{\text{int}}(e)([r \cdot |s\rangle \langle s|]) + \mathcal{Q}_{\text{int}}(f)([r \cdot |s\rangle \langle s|]) \\
&= \sum_{st \in S} \left(\sum_{k=1}^{\{\!\{e\}\!\}[t]} [r/\#_t \cdot |st\rangle \langle st|] + \sum_{k=1}^{\{\!\{f\}\!\}[t]} [r/\#_t \cdot |st\rangle \langle st|] \right) \\
&= \sum_{st \in S} \sum_{k=1}^{\{\!\{e+f\}\!\}[t]} [r/\#_t \cdot |st\rangle \langle st|].
\end{aligned}$$

Consider the case $e \cdot f$. For any $s \in S$ and $r \in \mathbb{R}$, by inductive hypotheses and Lemma 8.3.3.(ii),

$$\begin{aligned}
\mathcal{Q}_{\text{int}}(e \cdot f)([r \cdot |s\rangle \langle s|]) &= \mathcal{Q}_{\text{int}}(f)(\mathcal{Q}_{\text{int}}(e)([r \cdot |s\rangle \langle s|])) \\
&= \mathcal{Q}_{\text{int}}(f) \left(\sum_{st \in S} \sum_{k=1}^{\{\!\{e\}\!\}[t]} [r/\#_t \cdot |st\rangle \langle st|] \right) \\
&= \sum_{stw \in S} \sum_{k=1}^{\{\!\{e\}\!\}[t]} \sum_{l=1}^{\{\!\{f\}\!\}[w]} [r/(\#_t \cdot \#_w) \cdot |stw\rangle \langle stw|] \\
&= \sum_{st \in S} \sum_{k=1}^{\{\!\{e \cdot f\}\!\}[t]} [r/\#_t \cdot |st\rangle \langle st|].
\end{aligned}$$

Consider the case e^*. For any $s \in S$, by inductive hypothesis, Lemma 8.3.3.(ii) and the above proofs for $e + f$ and $e \cdot f$,

$$\begin{aligned}
\mathcal{Q}_{\text{int}}(e^*)([r \cdot |s\rangle \langle s|]) &= \mathcal{Q}_{\text{int}}(e)^*([r \cdot |s\rangle \langle s|]) = \sum_{i \geq 0} \mathcal{Q}_{\text{int}}(e)^i([r \cdot |s\rangle \langle s|]) \\
&= \sum_{i \geq 0} \mathcal{Q}_{\text{int}}(e^i)([r \cdot |s\rangle \langle s|]) = \sum_{i \geq 0} \sum_{st \in S} \sum_{k=1}^{\{\!\{e^i\}\!\}[t]} [r/\#_t \cdot |st\rangle \langle st|] \\
&= \sum_{st \in S} \sum_{k=1}^{\{\!\{e^*\}\!\}[t]} [r/\#_t \cdot |st\rangle \langle st|] \,.
\end{aligned}$$

□

This result indicates that equations of NKA are all possible tautologies when atomic symbols are interpreted as any (lifted) quantum superoperator. These equations and interpretations do not necessarily correspond to quantum programs, so further exploitation of algebraic structures specifically for quantum programs is possible.

Remark 8.4.1. *The completeness proof constructs interpretations with probabilistic processes only. It suggests that quantum processes have similar algebraic behaviors to probabilistic processes when probabilities are implicit (abstracted inside atomic operations). This is valid when measurements are not distinguished from other processes. We will discuss additional axioms for quantum measurements in Section 8.7.*

Most of the derived rules in our applications rely on external hypotheses aside from the NKA axioms. A formula with inequalities as hypotheses is called a Horn clause. We present the relation of the Horn theorems of NKA and quantum interpretations by the following theorem.

Corollary 8.4.3. *For expressions* $\{e_i\}_{i=1}^n, \{f_i\}_{i=1}^n \subset \mathrm{Exp}_\Sigma$ *and* $e, f \in \mathrm{Exp}_\Sigma$*, if*

$$\vdash_{\mathrm{NKA}} \left(\bigwedge_{i=1}^{n} e_i \leq f_i \right) \to e \leq f, \tag{8.4.1.3}$$

and $\mathrm{int} = (\mathcal{H}, \mathrm{eval})$ *satisfies* $\mathcal{Q}_{\mathrm{int}}(e_i) \preceq \mathcal{Q}_{\mathrm{int}}(f_i)$ *for* $1 \leq i \leq n$*, then* $\mathcal{Q}_{\mathrm{int}}(e) \preceq \mathcal{Q}_{\mathrm{int}}(f)$.

Note that the inequalities above can be replaced by equations, using the fact that $p = q \leftrightarrow p \leq q \wedge q \leq p$.

Proof. The proof comes from Theorem 8.3.8 similarly. Along the derivation of $e \leq f$, we apply the NKA axioms and premises $e_i \leq f_i$ for $1 \leq i \leq n$. The soundness of $e \leq f$ comes from the soundness of NKA axioms, proved in Theorem 8.3.8, and the soundness of each premises, provided by the assumption $\mathcal{Q}_{\mathrm{int}}(e_i) \preceq \mathcal{Q}_{\mathrm{int}}(f_i)$ for each $e_i \leq f_i$. □

8.4.2 Encoding of quantum programs

The syntax of a *quantum **while** program*, also called a program for simplicity, P is defined as follows. [1]

$$\begin{aligned} P \ ::= \ & \mathbf{skip} \mid \mathbf{abort} \mid q := |0\rangle \mid \overline{q} := U[\overline{q}] \mid P_1; P_2 \mid \\ & \mathbf{case}\ M[\overline{q}] \xrightarrow{i} P_i\ \mathbf{end} \mid \mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ P_1\ \mathbf{done}. \end{aligned}$$

[1] The **skip** statement does nothing and terminates. The **abort** statement announces that the program fails, and halts the program without any result. Statement $q := |0\rangle$ resets the register q to $|0\rangle$, and $\overline{q} := U[\overline{q}]$ applies a unitary operation on register set $\overline{q}$. These four statements' denotational semantics are called elementary superoperators. Note that there is no assignment statement due to the quantum no-cloning theorem [233]. The loop **while** $M[\overline{q}] = 1$ **do** P_1 **done** executes repeatedly. Each time it measures $\overline{q}$ by M. If the measurement result is 1, it executes P_1 and then starts over. Otherwise, it terminates. When there are only two branches, we define syntax sugar **if** $M[\overline{q}] = 1$ **then** P_1 **else** P_2 as an alternative to **case** $M[\overline{q}] \to^i P_i$ **end**. Moreover, if $P_2 \equiv$ **skip**, we write **if** $M[\overline{q}] = 1$ **then** P_1.

The denotational semantics of P is a quantum superoperator, denoted by $[\![P]\!]$. Ying [234] proves that:

$$[\![\mathbf{skip}]\!](\rho) = \rho, \quad \left[\!\!\left[\mathbf{case}\ M[\overline{q}] \xrightarrow{i} P_i\ \mathbf{end}\right]\!\!\right] = \sum_i \mathcal{M}_i \circ [\![P_i]\!],$$

$$[\![\mathbf{abort}]\!](\rho) = O_{\mathcal{H}}, \quad [\![q := |0\rangle]\!](\rho) = \sum_i |0\rangle_q \langle i| \rho |i\rangle_q \langle 0|,$$

$$[\![P_1; P_2]\!] = [\![P_1]\!] \circ [\![P_2]\!], \quad [\![\overline{q} := U[\overline{q}]]\!](\rho) = U_{\overline{q}} \rho U_{\overline{q}}^{\dagger},$$

$$[\![\mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ P\ \mathbf{done}]\!] = \sum_{n \geq 0} ((\mathcal{M}_1 \circ [\![P]\!])^n \circ \mathcal{M}_0),$$

where for a quantum measurement $\{M_i\}_{i \in I}$, $\mathcal{M}_i$ is defined by $\mathcal{M}_i(\rho) = M_i \rho M_i^{\dagger}$. Both $\circ$ and $\sum_i$ are operations over quantum superoperators.

We formally define how to encode a quantum program as an expression, and how to recover the denotational semantics of a quantum program from an expression.

Definition 8.4.4. *An* encoder setting *is a mapping E from a finite subset of $\mathcal{QC}(\mathcal{H})$ to Σ, that assigns a unique symbol in Σ to the elementary superoperators (qubit resetting, unitary application, and measurement branches) in the target programs.*

The encoder Enc *of a program to* Exp_{Σ} *with respect to an encoder setting E is defined*

inductively by:

$$\mathrm{Enc}(\mathbf{skip}) = 1; \qquad \mathrm{Enc}(q := |0\rangle) = E(\llbracket q := |0\rangle \rrbracket);$$

$$\mathrm{Enc}(\mathbf{abort}) = 0; \qquad \mathrm{Enc}(\overline{q} := U[\overline{q}]) = E(\llbracket \overline{q} := U[\overline{q}] \rrbracket);$$

$$\mathrm{Enc}(P_1; P_2) = \mathrm{Enc}(P_1) \cdot \mathrm{Enc}(P_2);$$

$$\mathrm{Enc}(\mathbf{case}\ M[\overline{q}] \xrightarrow{i} P_i\ \mathbf{end}) = \sum_i E(\mathcal{M}_i) \cdot \mathrm{Enc}(P_i);$$

$$\mathrm{Enc}(\mathbf{while}\, M[\overline{q}] = 1 \mathbf{do}\ P\, \mathbf{done}) = (E(\mathcal{M}_1) \cdot \mathrm{Enc}(P))^* \cdot E(\mathcal{M}_0),$$

where Σ_i in (8.4.2.1) is an abbreviation of expression summation.

Theorem 8.4.5. *For any quantum program P and encoder setting E, let $\mathrm{int} = (\mathcal{H}, E^{-1})$, where E^{-1} maps back the unique symbol for an elementary superoperator. Then*

$$\mathcal{Q}_{\mathrm{int}}(\mathrm{Enc}(P)) = \langle \llbracket P \rrbracket \rangle^{\uparrow}. \tag{8.4.2.1}$$

Proof of Theorem 8.4.5. We prove them by induction on P.

- For the base cases $P \equiv \mathbf{skip}, \mathbf{abort}$, the equation holds by definition. For $P \equiv q := |0\rangle$ and $\overline{q} := U[\overline{q}]$, we know $\mathrm{Enc}(P) \in \Sigma$ by the encoder setting E. With $E^{-1}(\mathrm{Enc}(P)) = \langle \llbracket P \rrbracket \rangle^{\uparrow}$, the equation holds.

- For $P = P_1; P_2$, by inductive hypotheses there are $\mathcal{Q}_{\mathrm{int}}(\mathrm{Enc}(P_1)) = \langle \llbracket P_1 \rrbracket \rangle^{\uparrow}$ and $\mathcal{Q}_{\mathrm{int}}(\mathrm{Enc}(P_2)) = \langle \llbracket P_2 \rrbracket \rangle^{\uparrow}$. Then by Lemma 8.3.10.(iii),

$$\mathcal{Q}_{\mathrm{int}}(\mathrm{Enc}(P)) = \mathcal{Q}_{\mathrm{int}}(\mathrm{Enc}(P_1)); \mathcal{Q}_{\mathrm{int}}(\mathrm{Enc}(P_2)) = \langle \llbracket P_1 \rrbracket \rangle^{\uparrow}; \langle \llbracket P_2 \rrbracket \rangle^{\uparrow} = \langle \llbracket P_1 \rrbracket \circ \llbracket P_2 \rrbracket \rangle^{\uparrow}.$$

- For $P \equiv \textbf{case } M[\overline{q}] \xrightarrow{i} P_i \textbf{ end}$, the inductive hypotheses are $\mathcal{Q}_{\text{int}}(\text{Enc}(P_i)) = \langle [\![P_i]\!] \rangle^{\uparrow}$. Then by Lemma 8.3.10.(iii),

$$\begin{aligned}\mathcal{Q}_{\text{int}}(\text{Enc}(P)) &= \sum_i (\mathcal{Q}_{\text{int}}(E(\mathcal{M}_i)); \mathcal{Q}_{\text{int}}(\text{Enc}(P_i)) \\ &= \sum_i (\langle \mathcal{M}_i \rangle^{\uparrow}; \langle [\![P_i]\!] \rangle^{\uparrow}) = \sum_i (\langle \mathcal{M}_i \circ [\![P_i]\!] \rangle^{\uparrow}) = \langle \sum_i (\mathcal{M}_i \circ [\![P_i]\!]) \rangle^{\uparrow}.\end{aligned}$$

- For $P \equiv \textbf{while } M[\overline{q}] = 1 \textbf{ do } S \textbf{ done}$, the inductive hypothesis becomes $\mathcal{Q}_{\text{int}}(\text{Enc}(S)) = \langle [\![S]\!] \rangle^{\uparrow}$. By [234] $\sum_{n \geq 0}((\mathcal{M}_1 \circ [\![S]\!])^n \circ \mathcal{M}_0)$ exists in $\mathcal{QC}(\mathcal{H})$, so by Lemma 8.3.10.(iii) and linearity of transformations in $\mathcal{P}(\mathcal{H})$,

$$\begin{aligned}\mathcal{Q}_{\text{int}}(\text{Enc}(P)) &= (\mathcal{Q}_{\text{int}}(E(\mathcal{M}_1)); \mathcal{Q}_{\text{int}}(\text{Enc}(S)))^* \mathcal{Q}_{\text{int}}(E(\mathcal{M}_0)) \\ &= (\langle \mathcal{M}_1 \rangle^{\uparrow}; \langle [\![S]\!] \rangle^{\uparrow})^*; \langle \mathcal{M}_0 \rangle^{\uparrow} = \left(\sum_{n \geq 0} (\langle \mathcal{M}_1 \rangle^{\uparrow}; \langle [\![S]\!] \rangle^{\uparrow})^n \right); \langle \mathcal{M}_0 \rangle^{\uparrow} \\ &= \sum_{n \geq 0} ((\langle \mathcal{M}_1 \rangle^{\uparrow}; \langle [\![S]\!] \rangle^{\uparrow})^n; \langle \mathcal{M}_0 \rangle^{\uparrow}) = \sum_{n \geq 0} \langle (\mathcal{M}_1 \circ [\![S]\!])^n \circ \mathcal{M}_0 \rangle^{\uparrow} \\ &= \langle \sum_{n \geq 0} ((\mathcal{M}_1 \circ [\![S]\!])^n \circ \mathcal{M}_0) \rangle^{\uparrow}.\end{aligned}$$

□

Note that in real applications, we usually define the encoder setting E jointly for multiple programs $\{P_i\}$ for technical convenience and easy comparison.

Now we have all the ingredients for Theorem 8.1.1.

Proof of Theorem 8.1.1. We have constructed the quantum path model and proved it a sound model of NKA in Theorem 8.3.8, leading to the soundness of Horn theorems by Corollary 8.4.3.

We also show an embedding of quantum superoperators into quantum path actions in Lemma 8.3.10.(ii), so Horn theorems are interpreted as quantum superoperator equivalences. For each quantum program, we encode it with a symbolic expression whose interpretation corresponds to its denotational semantics, according to Theorem 8.4.5. Hence, if the NKA equivalence of quantum programs' encoding is derivable, the equivalence of their denotational semantics is induced. □

In the next sections, we show applications of Theorem 8.1.1.

8.5 Validation of quantum compiler optimizing rules

We demonstrate a few quantum compiler optimizing rules and their validation in NKA, in light of a similar application of KAT [179]. Note that many classical compiler optimizing rules do not hold or make sense in the quantum setting. We have carefully selected those rules with reasonable quantum counterparts, as well as quantum-specific rules found in real quantum applications.

The validation of quantum program equivalence via NKA consists of three steps: (1) *program encoding*: encode the programs as expressions over an alphabet; (2) *condition formulation*: identify necessary hypotheses and construct a formula that encodes hypotheses and target equation; (3) *NKA derivation*: derive the formula with the NKA axioms.

8.5.1 Loop unrolling

Consider programs UNROLLING1 and UNROLLING2 in Figure 8.3 with a program P and a projective measurement M.

Program Encoding: We encode the two programs by expressions $\mathrm{Enc}(\textsc{Unrolling1}) = (m_0 p)^* m_1$

and $\mathrm{Enc}(\textsc{Unrolling2}) = (m_0 p(m_0 p + m_1 \cdot 1))^* m_1$. The encoder setting is inferred easily.

Condition Formulation: Because M is a projective measurement, $\mathcal{M}_1 \circ \mathcal{M}_1 = \mathcal{M}_1$ and $\mathcal{M}_1 \circ \mathcal{M}_0 = \mathcal{O}_{\mathcal{H}}$ can be encoded by $m_1 m_1 = m_1$ and $m_1 m_0 = 0$. Their equivalence can be verified by the following formula:

$$\begin{aligned} &\vdash_{\text{NKA}} \; m_1 m_1 = m_1 \wedge m_1 m_0 = 0 \rightarrow \\ &\qquad (m_0 p)^* m_1 = (m_0 p(m_0 p + m_1 \cdot 1))^* m_1. \end{aligned} \tag{8.5.1.1}$$

NKA Derivation: This formula can be derived in NKA by:

$$\begin{aligned}
&(m_0 p(m_0 p + m_1 \cdot 1))^* m_1 \\
&= (m_0 p m_0 p + m_0 p m_1)^* m_1 && \text{(distributive-law)} \\
&= (m_0 p m_0 p)^* (m_0 p m_1 (m_0 p m_0 p)^*)^* m_1 && \text{(denesting)} \\
&= (m_0 p m_0 p)^* (m_0 p m_1 (1 + m_0 p m_0 p (m_0 p m_0 p)^*))^* m_1 && \text{(fixed-point)} \\
&= (m_0 p m_0 p)^* (m_0 p m_1)^* m_1 && (m_1 m_0 = 0) \\
&= (m_0 p m_0 p)^* (1 + m_0 p m_1 (1 + m_0 p m_1 (m_0 p m_1)^*)) m_1 && \text{(fixed-point)} \\
&= (m_0 p m_0 p)^* (1 + m_0 p m_1) m_1 && (m_1 m_0 = 0) \\
&= (m_0 p m_0 p)^* (1 + m_0 p) m_1 && (m_1 m_1 = m_1, \text{ distributive-law}) \\
&= (m_0 p)^* m_1. && \text{(unrolling)}
\end{aligned}$$

By Theorem 8.1.1, we have $[\![\textsc{Unrolling1}]\!] = [\![\textsc{Unrolling2}]\!]$.

UNROLLING1 $\equiv$	UNROLLING2 $\equiv$
while $M[q] = 0$ **do**	**while** $M[q] = 0$ **do**
$\quad P$	$\quad P$; **if** $M[q] = 0$ **then** P
done.	**done**.
BOUNDARY1 $\equiv$	BOUNDARY2 $\equiv$
while $M[w] = 0$ **do**	$q := U[q]$;
$\quad q := U[q]$;	**while** $M[w] = 0$ **do**
$\quad P$;	$\quad P$;
$\quad q := U^{-1}[q]$	**done**;
done.	$q := U^{-1}[q]$.

Figure 8.3: Two pairs of equivalent programs with conditions.

8.5.2 Loop boundary

This rule is quantum-specific because it makes use of the reversible property of quantum operations. Consider programs BOUNDARY1 and BOUNDARY2 in Figure 8.3, where P is an arbitrary program. Here the unitary U acting on q does not affect the measurement on qubit w. In other words, quantum measurement M_0 and M_1 commute with U.

Program Encoding: We encode these program by expressions $\mathrm{Enc}(\text{BOUNDARY1}) = (m_0 upu^{-1})^* m_1$ and $\mathrm{Enc}(\text{BOUNDARY2}) = u(m_0 p)^* m_1 u^{-1}$, where the encoder setting E can be inferred.

Condition Formulation: The reversibility property $UU^{-1} = U^{-1}U = I$ can be encoded by $uu^{-1} = u^{-1}u = 1$ (at the level of quantum superoperators). Besides, the commutativity property of measurement and U is encoded as $um_0 = m_0 u$ and $um_1 = m_1 u$. Then the formula we need

to derive is

$$\vdash_{\text{NKA}} \; uu^{-1} = u^{-1}u = 1 \wedge um_0 = m_0u \wedge um_1 = m_1u \rightarrow$$
$$(m_0upu^{-1})^*m_1 = u(m_0p)^*m_1u^{-1}. \tag{8.5.2.1}$$

NKA Derivation: The derivation of this formula in NKA is

$$\begin{aligned}
(m_0upu^{-1})^*m_1 &= (um_0pu^{-1})^*m_1 && (um_0 = m_0u)\\
&= (1 + u(m_0pu^{-1}u)^*m_0pu^{-1})m_1 && (\text{product-star})\\
&= u(m_0p)^*m_1u^{-1}. && (u^{-1}u = 1, \text{fixed-point})
\end{aligned}$$

Then $[\![\text{BOUNDARY1}]\!] = [\![\text{BOUNDARY2}]\!]$ by Theorem 8.1.1.

8.5.3 Optimizing quantum signal processing

We showcase the use of the Loop Boundary rule to optimize, as observed in [235], one leading quantum Hamiltonian simulation algorithm called quantum signal processing (QSP) [236], as well as its algebraic verification.

The QSP implementation before (QSP) and after (QSP') the optimization is illustrated in Figure 8.4. The algorithm QSP simulates the Hamiltonian $H = \sum_{l=1}^{L} \alpha_l H_l$ on qubit register q with high probability. Let us explain the components in QSP briefly, whose details imply some commutativity conditions for our purpose. $|G\rangle = 1/\sqrt{\sum_{l=1}^{L} \alpha_l} \sum_{l=1}^{L} \sqrt{\alpha_l}\, |l\rangle$ is a state defined by H. $\Phi = \sum_{j=1}^{n} |j\rangle\langle j| \otimes e^{-i\phi_j \sigma^Z/2}$ is an operation that rotates qubit p with a pre-defined angle ϕ_j when the control quantum register c is j. Unitary $S = (1-i)\,|G\rangle\langle G| - I$ is a partial reflection

$$
\begin{array}{lll}
\textsc{qsp}[q] \equiv & c := |n\rangle\,;\ p := |+\rangle\,;\ r := |G\rangle\,; & (c_0 p_0 r_0) \\
& \textbf{while } M[c] = 1 \textbf{ do} & (\{m_i\}) \\
& \quad c, p := \Phi[c, p]; & (\phi) \\
& \quad r := S[r]; & (s) \\
& \quad p, r, q := \mathrm{C}_W[p, r, q]; & (w_c) \\
& \quad r := S^{-1}[r]; & (s^{-1}) \\
& \quad c, p := \Phi^{-1}[c, p]; & (\phi^{-1}) \\
& \quad c := \mathrm{Dec}[c] & (d) \\
& \textbf{done}; & \\
& \textbf{if } M_{|+\rangle|G\rangle}[p, r] = 0 & \\
& \textbf{then abort} & (\tau_0 0 + \tau_1 1)
\end{array}
$$

$$
\begin{array}{lll}
\textsc{qsp'}[q] \equiv & c := |n\rangle\,;\ p := |+\rangle\,;\ r := |G\rangle\,; & (c_0 p_0 r_0) \\
& \textbf{while } M[c] = 1 \textbf{ do} & (\{m_i\}) \\
& \quad c, p := \Phi[c, p]; & (\phi) \\
& \quad p, r, q := \mathrm{C}_W[p, r, q]; & (w_c) \\
& \quad c, p := \Phi^{-1}[c, p]; & (\phi^{-1}) \\
& \quad c := \mathrm{Dec}[c] & (d) \\
& \textbf{done}; & \\
& \textbf{if } M_{|+\rangle|G\rangle}[p, r] = 0 & \\
& \textbf{then abort} & (\tau_0 0 + \tau_1 1)
\end{array}
$$

Figure 8.4: The program QSP and QSP'. The measurement $M[c]$ is $\{M_0 = I_c - |0\rangle\langle 0|, M_1 = |0\rangle\langle 0|\}$ on register c. The measurement $M_{|+\rangle|G\rangle}[p, r]$ is $\{M_0 = I_{p,r} - |+\rangle\langle +| \otimes |G\rangle\langle G|, M_1 = |+\rangle\langle +| \otimes |G\rangle\langle G|\}$ on register p and r jointly.

operator about state $|G\rangle$, and $W = -i((2|G\rangle\langle G| - I) \otimes I)\sum_{l=1}^{L} |l\rangle\langle l| \otimes H_l$, which defines $\mathrm{C}_W = |+\rangle\langle +| \otimes I + |-\rangle\langle -| \otimes W$. $\mathrm{Dec} = |n\rangle\langle 0| + \sum_{j=1}^{n} |j-1\rangle\langle j|$ is the unitary implementing $j \mapsto (j-1) \mod n$.

Program Encoding: We encode the programs in Figure 8.4 as

$$\mathrm{Enc}(\textsc{qsp}) = c_0 p_0 r_0 (m_1 \varphi s w_c s^{-1} \varphi^{-1} d)^* m_0 (\tau_0 0 + \tau_1 1),$$

$$\mathrm{Enc}(\textsc{qsp'}) = c_0 p_0 r_0 (m_1 \varphi w_c \varphi^{-1} d)^* m_0 (\tau_0 0 + \tau_1 1).$$

The detailed encoder setting is self-explanatory.

Condition Formulation: One can derive commutative conditions because $c, p := \Phi[c, p]$ and $r := S[r]$, similarly $r := S^{-1}[r]$ and $c, p := \Phi^{-1}[c, p]; c := \mathrm{Dec}[c]$, apply on different quantum variables and hence commute. Algebraically, we hence have $\varphi s = s\varphi$, and $\varphi^{-1} d s^{-1} =$

$s^{-1}\varphi^{-1}d$. Moreover, $M[c]$ is commutable to $r := S[r]$, so $m_1 s = s m_1$ and $m_0 s = s m_0$. Since $S\left|G\right\rangle\left\langle G\right|S^\dagger = \left|G\right\rangle\left\langle G\right|$, we have $r_0 s = r_0$. Similarly the Kraus operator $(\left|+\right\rangle\left\langle +\right| \otimes \left|G\right\rangle\left\langle G\right|) \cdot (I_p \otimes ((1+i)\left|G\right\rangle\left\langle G\right| - I_r)) = i\left|+\right\rangle\left\langle +\right| \otimes \left|G\right\rangle\left\langle G\right|$, and the phases are cancelled when represented by superoperator. This is encoded as $s^{-1}\tau_1 = \tau_1$. Then we need to show $\mathrm{Enc}(\textsc{qsp}) = \mathrm{Enc}(\textsc{qsp'})$ with these hypotheses and the NKA axioms.

NKA derivation: By (8.5.2.1), we have

$$
\begin{aligned}
& c_0 p_0 r_0 (m_1 \varphi s w_c s^{-1} \varphi^{-1} d)^* m_0 (\tau_0 0 + \tau_1 1) \\
= {} & c_0 p_0 r_0 (s m_1 \varphi w_c \varphi^{-1} d s^{-1})^* m_0 \tau_1 && \text{(commutativity)} \\
= {} & c_0 p_0 r_0 s (m_1 \varphi w_c \varphi^{-1} d)^* m_0 s^{-1} \tau_1 && \text{((8.5.2.1))} \\
= {} & c_0 p_0 r_0 (m_1 \varphi w_c \varphi^{-1} d)^* m_0 \tau_1, && \text{(absorption-hypotheses)} \\
= {} & c_0 p_0 r_0 (m_1 \varphi w_c \varphi^{-1} d)^* m_0 (\tau_0 0 + \tau_1 1).
\end{aligned}
$$

Notice that m_1 and φ do not commute, so we cannot apply (8.5.2.1) further. By Corollary 8.4.3, Theorem 8.4.5 and Lemma 8.3.10.(ii), $[\![\textsc{qsp}]\!] = [\![\textsc{qsp'}]\!]$. Note that in $\textsc{qsp'}$, S and S^{-1} vanish, which could largely reduce the total gate count.

8.6 Normal form theorem of quantum while programs

Here we use NKA to prove a quantum counterpart of the classic Böhm-Jacopini theorem [237], namely, a normal form of quantum **while** programs consisting of only a single loop. The normal form of classical programs depends on the folk operation, which copies the value of a variable to a new variable. However, in quantum programs, the no-cloning theorem prevents us

from directly copying unknown states. Our approach is to store every measurement result in an augmented classical space and depends on the classical variables to manipulate the control flow of the program. We note a quantum version of the Böhm-Jacopini theorem was recently shown in [238], however, using a completely different and non-algebraic approach.

Let us illustrate our idea with a simple example below first. To unify the two **while** loops of ORIGINAL into one, we redesign the control flow as in CONSTRUCTED with a fresh classical guard variable $g \in \{0, 1, 2\}$.

ORIGINAL $\equiv$	CONSTRUCTED $\equiv$
while $M_1[p] = 1$ **do** P_1 **done**;	$g := \lvert 1\rangle$;
while $M_2[p] = 1$ **do** P_2 **done**;	**while** Meas$[g] > 0$ **do**
$g := \lvert 0\rangle$,	**if** Meas$[g] > 1$ **then**
	if $M_2[p] = 1$ **then** P_2 **else** $g := \lvert 0\rangle$
	else
	if $M_1[p] = 1$ **then** P_1 **else** $g := \lvert 2\rangle$
	done

Here Meas$[g]$ is the computational basis measurement on variable g. When g is classical, Meas$[g]$ returns the value of g, and does not modify g. The variable g is used to store the measurement results and decide which branch the program executes in the next round. We prove $[\![\text{ORIGINAL}]\!] = [\![\text{CONSTRUCTED}]\!]$ via NKA, using the outline in Section 8.5.

Program Encoding: We encode $g := \lvert i\rangle$ as g^i, and Meas$[g] > i$ as $g_{>i}$ and $g_{\leq i}$. Then the two

programs are encoded as

$$\mathrm{Enc}(\textsc{Original}) = (m_{11}p_1)^* m_{10}(m_{21}p_2)^* m_{20}g^0,$$

$$\mathrm{Enc}(\textsc{Constructed}) = g^1(g_{>0}(g_{>1}(m_{21}p_2 + m_{20}g^0) + g_{\leq 1}(m_{11}p_1 + m_{10}g^2)))^* g_{\leq 0}.$$

Condition Formulation: Since g is fresh, operations on g commutes with the quantum measurements M_1, M_2 and subprograms P_1, P_2. This is encoded as $g^i m_{jk} = m_{jk} g^i, g^i p_j = p_j g^i$. Two consecutive assignment on g will make the first one be overwritten, which is encoded as $g^i g^j = g^j$. On top of these, $g^i g_{>j} = g^i$ if $i > j$ and $g^i g_{>j} = 0$ if $i \leq j$. Similarly, $g^i g_{\leq j} = g^i$ if $i \leq j$ and $g^i g_{\leq j} = 0$ if $i > j$.

NKA derivation: To simplify the proof, let $X = g_{>0}g_{>1}(m_{21}p_2 + m_{20}g^0), Y = g_{>0}g_{\leq 1}(m_{11}p_1 + m_{10}g^2)$. Then $\mathrm{Enc}(\textsc{Constructed})$ is equivalent to $g^1(X+Y)^* g_{\leq 0}$. We simplify $g^i X^*$ first.

$$\begin{aligned}
g^1 X^* &= g^1(1 + g_{>0}g_{>1}(m_{20}g^0 + m_{21}p_2)X^*) = g^1 && \text{(fixed-point, distributive-law)}\\
g^2 X^* &= g^2(g_{>0}g_{>1}m_{21}p_2)^*(g_{>0}g_{>1}m_{20}g^0 \\
&\qquad \cdot (1 + g_{>0}g_{>1}m_{21}p_2(g_{>0}g_{>1}m_{21}p_2)^*))^* && \text{(denesting, fixed-point)}\\
&= (m_{21}p_2)^* g^2(g_{>0}g_{>1}m_{20}g^0)^* && \text{(star-rewrite)}\\
&= (m_{21}p_2)^* g^2(1 + g_{>0}g_{>1}m_{20}g^0) && \text{(fixed-point)}\\
&= (m_{21}p_2)^*(g^2 + m_{20}g^0). && \text{(distributive-law)}
\end{aligned}$$

Consider $g^1(X+Y)^* = g^1X^*(YX^*)^* = g^1(YX^*)^*$, and then

$$\begin{aligned}
g^1(YX^*)^* &= g^1(g_{>0}g_{\leq 1}m_{11}p_1X^*)^*(g_{>0}g_{\leq 1}m_{10}g^2X^*(g_{>0}g_{\leq 1}m_{11}p_1X^*)^*)^* && \text{(denesting)} \\
&= (m_{11}p_1)^*g^1(g_{>0}g_{\leq 1}m_{10}m_{21}^*(g^2+m_{20}g^0) \\
&\qquad \cdot (1+(g_{>0}g_{\leq 1}m_{11}p_1X^*)(g_{>0}g_{\leq 1}m_{11}p_1X^*)^*)) && \text{(star-rewrite, fixed-point)} \\
&= (m_{11}p_1)^*m_{10}(m_{21}p_2)^*(g^2+m_{20}g^0).
\end{aligned}$$

Insert the above equation into $g^1(X+Y)^*g_{\leq 0}$.

$$g^1(X+Y)^*g_{\leq 0} = (m_{11}p_1)^*m_{10}(m_{21}p_2)^*(g^2+m_{20}g^0)g_{\leq 0}(m_{11}p_1)^*m_{10}(m_{21}p_2)^*m_{20}g^0.$$

This is exactly $\mathrm{Enc}(\textsc{Constructed}) = \mathrm{Enc}(\textsc{Original})$. Applying the main Theorem 8.1.1 gives $[\![\textsc{Constructed}]\!] = [\![\textsc{Original}]\!]$. Hence the two loops have been merged into one, with an additional classical space restored to 0 at the end.

We employ a similar idea to arbitrary programs by induction. Note that our above example corresponds to the case $S_1; S_2$ in induction. Our analysis above, which results in an equivalent program with one **while**-loop and additional classical space, constitutes proof in that case. The more complicated cases are proved similarly.

Theorem 8.6.1. *For any quantum while program P over Hilbert space $\mathcal{H}$, there are a classical space $\mathcal{C}$ and a quantum while program*

$$P_0;\ \textbf{while}\ M\ \textbf{do}\ P_1\ \textbf{done};\ p_{\mathcal{C}} := |0\rangle \tag{8.6.0.1}$$

equivalent to P; $p_{\mathcal{C}} := |0\rangle$ over $\mathcal{H} \otimes \mathcal{C}$, where P_0, P_1 are while-free, $p_{\mathcal{C}} := |0\rangle$ resets the classical variables in $\mathcal{C}$ to $|0\rangle$.

Proof of Theorem 8.6.1. We prove the normal form theorem by induction on the program P. For each step we introduce a classical guard variable g whose value is limited in a finite set $\{0, 1, ..., n-1\}$, and denote the space of g by $\mathcal{C}_n$. We encode $g := |i\rangle$ as g^i, the measurement $\mathrm{Meas}[g] = i$ as g_i and the reset of space $\mathcal{C}$ as c. Each time g is independent of the existing space, so the following assumptions hold for any i, j in the value set:

- g^i commutes with every elements except for g^j.
- $g^i g_j = \delta_{ij} g^i$, where $\delta_{ij} = 1$ when $i = j$, and $\delta_{ij} = 0$ when $i \neq j$.
- $g^i g^j = g^j$.

(a) For the base case where $P = \mathbf{skip} \mid \mathbf{abort} \mid q := |0\rangle \mid \overline{q} = U[\overline{q}]$, they are while-free. Let $\mathcal{C} = \mathcal{C}_1$ the space with only one value. We claim $P; g := |0\rangle; \mathbf{while}\ \mathrm{Meas}[g] = 1\ \mathbf{do\ skip\ done}; g := |0\rangle$ is equivalent to $P; g := |0\rangle$. The NKA encoding of these two programs are $pg^0(g_1 1)^* g_0 g^0$ and pg^0. This motivates the following derivation:

$$g^0(g_1 1)^* g_0 = g^0 g_0 + g^0 g_1 g_1^* g_0 = g^0.$$

Hence $pg^0(g_1 1)^* g_0 g^0 = pg^0 g^0 = pg^0$.

(b) For the $S_1; S_2$ case, by inductive hypothesis we have two external space $\mathcal{C}^1$ and $\mathcal{C}^2$ such that $S_i; p_{\mathcal{C}^i} := |0\rangle$ is equivalent to $P_{i0}; \mathbf{while}\ M_i\ \mathbf{do}\ P_{i1}\ \mathbf{done}; p_{\mathcal{C}^i} := |0\rangle$, where P_{ij} is

while-free. We claim $S_1; S_2; p_{\mathcal{C}^1 \otimes \mathcal{C}^2 \otimes \mathcal{C}_3} := |0\rangle$ and

$$
\begin{aligned}
&P_{10}; g := |1\rangle\,; \\
&\textbf{while } \mathrm{Meas}[g] > 0 \textbf{ do} \\
&\quad \textbf{if } \mathrm{Meas}[g] = 1 \textbf{ then} \\
&\qquad \textbf{if } M_1 \textbf{ then } P_{11} \\
&\qquad \textbf{else } P_{20}; g := |2\rangle \\
&\quad \textbf{else} \\
&\qquad \textbf{if } M_2 \textbf{ then } P_{21} \\
&\qquad \textbf{else } g := |0\rangle \\
&\textbf{done}; \\
&p_{\mathcal{C}^1 \otimes \mathcal{C}^2 \otimes \mathcal{C}_3} := |0\rangle\,,
\end{aligned}
$$

are equivalent, whose encodings are $s_1 s_2 c_1 c_2 g^0$ and $p_{10} g^1((g_1 + g_2)(g_1(m_{11} p_{11} + m_{12} p_{20} g^2) + (g_0 + g_2)(m_{21} p_{21} + m_{22} g^0)))^* g_0 c_1 c_2 g^0$.

Notice that c_1 acts on $\mathcal{C}^1$, so c_1 is commutable to those operators acting on $\mathcal{H} \otimes \mathcal{C}^2 \otimes \mathcal{C}_3$. By inductive hypothesis, there is $s_i c_i = p_{i0}(m_{i1} p_{i1})^* m_{i2} c_i$, so

$$
\begin{aligned}
s_1 s_2 c_1 c_2 g^0 &= s_1 c_1 s_2 c_2 g^0 \\
&= p_{10}(m_{11} p_{11})^* m_{12} c_1 p_{20}(m_{21} p_{21})^* m_{22} c_2 g^0 \\
&= p_{10}(m_{11} p_{11})^* m_{12} p_{20}(m_{21} p_{21})^* m_{22} c_1 c_2 g^0.
\end{aligned}
$$

Let $X = (g_1 + g_2)(g_1(m_{11}p_{11} + m_{12}p_{20}g^2) + (g_0 + g_2)(m_{21}p_{21} + m_{22}g^0)) = g_1(m_{11}p_{11} + m_{12}p_{20}g^2) + g_2(m_{21}p_{21} + m_{22}g^0)$, and $Y = g_1(m_{11}p_{11} + m_{12}p_{20}g^2)$. Then by denesting rule:

$$
\begin{aligned}
g^1X^* &= g^1(g_1(m_{11}p_{11} + m_{12}p_{20}g^2))^*(g_2(m_{21}p_{21} + m_{22}g^0)(g_1(m_{11}p_{11} + m_{12}p_{20}g^2))^*)^* \\
&= g^1Y^*(g_2(m_{21}p_{21} + m_{22}g^0)Y^*)^*. \\
g^1Y^* &= g^1(g_1m_{11}p_{11})^*(g_1m_{12}p_{20}g^2(g_1m_{11}p_{11})^*)^* \\
&= (m_{11}p_{11})^*g^1(g_1m_{12}p_{20}g^2 + g_1m_{12}p_{20}g^2g_1m_{11}p_{11}(g_1m_{11}p_{11})^*))^* \\
&= (m_{11}p_{11})^*g^1(g_1m_{12}p_{20}g^2)^* \\
&= (m_{11}p_{11})^*g^1(1 + g_1m_{12}p_{20}g^2 + g_1m_{12}p_{20}g^2g_1m_{12}p_{20}g^2(g_1m_{12}p_{20}g^2)^*) \\
&= (m_{11}p_{11})^*(g^1 + m_{12}p_{20}g^2). \\
g^2Y^* &= g^2.
\end{aligned}
$$

By star-rewrite , we have:

$$
\begin{aligned}
& g^2(g_2(m_{21}p_{21} + m_{22}g^0)Y^*)^* \\
&= g^2(g_2m_{21}p_{21}Y^*)^*(g_2m_{22}g^0Y^*(g_2m_{21}p_{21}Y^*)^*)^* \\
&= g^2(g_2m_{21}p_{21})^*(g_2m_{22}g^0 + g_2m_{22}g^0g_2m_{21}p_{21}Y^*(g_2m_{21}p_{21}Y^*)^*) \\
&= (m_{21}p_{21})^*g^2(g_2m_{22}g^0)^* \\
&= (m_{21}p_{21})^*g^2(1 + g_2m_{22}g^0 + g_2m_{22}g^0(g_2m_{22}g^0)^*) \\
&= (m_{21}p_{21})^*(g^2 + m_{22}g^0).
\end{aligned}
$$

Hence we have:

$$
\begin{aligned}
& p_{10}g^1((g_1+g_2)(g_1(m_{11}p_{11}+m_{12}p_{20}g^2)+(g_0+g_2)(m_{21}p_{21}+m_{22}g^0)))^*g_0c_1c_2g^0 \\
&= p_{10}(m_{11}p_{11})^*(g^1+m_{12}p_{20}g^2)(g_2(m_{21}p_{21}+m_{22}g^0)Y^*)^*g_0c_1c_2g^0 \\
&= p_{10}(m_{11}p_{11})^*g^1g_0c_1c_2g^0+p_{10}(m_{11}p_{11})^*m_{12}p_{20}g^2(g_2(m_{21}p_{21}+m_{22}g^0)Y^*)^*g_0c_1c_2g^0 \\
&= p_{10}(m_{11}p_{11})^*m_{12}p_{20}(m_{21}p_{21})^*(g^2+m_{22}g^0)g_0c_1c_2g^0 \\
&= p_{10}(m_{11}p_{11})^*m_{12}p_{20}(m_{21}p_{21})^*m_{22}c_1c_2g^0 \\
&= s_1s_2c_1c_2g^0.
\end{aligned}
$$

(c) For the **case** $M \xrightarrow{i} S_i$ **end** case, w.l.o.g. we assume the measurement results are $\{1, 2, ..., n\}$. By inductive hypothesis we have two external spaces $\{\mathcal{C}^i\}_{1\le i\le n}$ such that $S_i; p_{\mathcal{C}^i} := |0\rangle$ is equivalent to $P_{i0}; \textbf{while } M_i \textbf{ do } P_{i1} \textbf{ done}; p_{\mathcal{C}^i} := |0\rangle$, where P_{ij} is while-free. Let $\mathcal{C} = \left(\bigotimes_{1\le i\le n}\mathcal{C}^i\right)\otimes\mathcal{C}_{n+1}$. We claim **case** $M \xrightarrow{i} S_i$ **end**; $p_C = |0\rangle$ and

$$
\begin{aligned}
&\textbf{case } M \xrightarrow{i} P_{i0}; g := |i\rangle \textbf{ end} \\
&\textbf{while } \mathrm{Meas}[g] > 0 \textbf{ do} \\
&\quad \textbf{case } \mathrm{Meas}[g] \xrightarrow{i>0} \\
&\qquad \textbf{if } M_i \textbf{ then } P_{i1} \textbf{ else } g := |0\rangle \\
&\quad \textbf{end} \\
&\textbf{done}; \\
&p_{\mathcal{C}} := |0\rangle
\end{aligned}
$$

are equivalent, whose encodings are $(\sum_{i=1}^{n} m_i s_i)(\prod_{i=1}^{n} c_i)g^0$ and

$$\left(\sum_{i=1}^{n} m_i p_{i0} g^i\right)\left(\left(\sum_{i=1}^{n} g_i\right)\left(\sum_{i=1}^{n} g_i(m_{i1}p_{i1} + m_{i2}g^0)\right)\right)^* g_0 \left(\prod_{i=1}^{n} c_i\right) g^0.$$

First we show $\textbf{case } M \xrightarrow{i} S_i \textbf{ end}; p_C = |0\rangle$ is equivalent to

$$\textbf{case } M \xrightarrow{i} S_i; p_{C^i} := |0\rangle \textbf{ end};$$

$$p_C = |0\rangle\,.$$

$\left(\sum_{1\le i\le n} m_i s_i\right)\left(\prod_{i=1}^{n} c_i\right) g^0 = \left(\sum_{1\le i\le n} m_i s_i c_i\right)\left(\prod_{i=1}^{n} c_i\right) g^0$ is what we need to derive. Because each $\mathcal{C}^i$ and C_3 are disjoint, we have c_i commutes each other for $1 \le i \le n$, and $c_i c_i = c_i$. With these assumptions added, the two expressions are equivalent by distributive law.

Then we could apply inductive hypothesis $p_{i0}(m_{i1}p_{i1})^* m_{i2}c_i = s_i c_i$ on each branch. Let $X = (\sum_{i=1}^{n} g_i)\left(\sum_{i=1}^{n} g_i(m_{i1}p_{i1} + m_{i2}g^0)\right) = \sum_{i=1}^{n} g_i(m_{i1}p_{i1} + m_{i2}g^0), Y_i = g_i m_{i1} p_{i1} + g_i m_{i2} g^0$ for convenience. By denesting rule, $g^i X^* = g^i Y_i^* \left(\left(\sum_{j\neq i} g_j(m_{j1}p_{j1} + m_{j2}g^0)\right) Y_i^*\right)^*$. Notice that for $1 \le i \le n$,

$$\begin{aligned}
g^i Y_i^* &= g^i (g_i m_{i1} p_{i1})^* (g_i m_{i2} g^0 (g_i m_{i1} p_{i1})^*)^* \\
&= (m_{i1}p_{i1})^* g^i (g_i m_{i2} g^0 + g_i m_{i2} g^0 g_i m_{i1} p_{i1} (g_i m_{i1} p_{i1})^*)^* \\
&= (m_{i1}p_{i1})^* g^i (g_i m_{i2} g^0)^* \\
&= (m_{i1}p_{i1})^* g^i (1 + g_i m_{i2} g^0 + g_i m_{i2} g^0 g_i m_{i2} g^0 (g_i m_{i2} g^0)^*) \\
&= (m_{i1}p_{i1})^* (g^i + m_{i2} g^0).
\end{aligned}$$

Meanwhile, for all $1 \leq i \leq n$, $g^0 \left(\left(\sum_{j \neq i} g_j (m_{j1} p_{j1} + m_{j2} g^0) \right) Y_i^* \right)^* = g^0$, and $g^i = g^i \left(\left(\sum_{j \neq i} g_j (m_{j1} p_{j1} + m_{j2} g^0) \right) Y_i^* \right)^*$.

Combining them up results in $g^i X^* = (m_{i1} p_{i1})^* (g^i + m_{i2} g^0)$ for $1 \leq i \leq n$. Thus

$$
\begin{aligned}
& \left(\sum_{i=1}^{n} m_i p_{i0} g^i \right) \left(\left(\sum_{i=1}^{n} g_i \right) \left(\sum_{i=1}^{n} g_i (m_{i1} p_{i1} + m_{i2} g^0) \right) \right)^* g_0 \left(\prod_{i=1}^{n} c_i \right) g^0 \\
= & \left(\sum_{i=1}^{n} m_i p_{i0} g^i X^* \right) g_0 \left(\prod_{i=1}^{n} c_i \right) g^0 \\
= & \left(\sum_{i=1}^{n} m_i p_{i0} (m_{i1} p_{i1})^* (g^i + m_{i2} g^0) \right) g_0 \left(\prod_{i=1}^{n} c_i \right) g^0 \\
= & \left(\sum_{i=1}^{n} m_i p_{i0} (m_{i1} p_{i1})^* m_{i2} g^0 \right) g_0 \left(\prod_{i=1}^{n} c_i \right) g^0 \\
= & \left(\sum_{i=1}^{n} m_i p_{i0} (m_{i1} p_{i1})^* m_{i2} c_i \right) \left(\prod_{i=1}^{n} c_i \right) g^0 \\
= & \left(\sum_{i=1}^{n} m_i s_i c_i \right) \left(\prod_{i=1}^{n} c_i \right) g^0 \\
= & \left(\sum_{i=1}^{n} m_i s_i \right) \left(\prod_{i=1}^{n} c_i \right) g^0.
\end{aligned}
$$

(d) For the **while** M_1 **do** S **done** case, by inductive hypothesis we have $\mathcal{C}$ such that $S; p_{\mathcal{C}} := |0\rangle$ is equivalent to $P_1;$ **while** M_2 **do** P_2 **done**$; p_{\mathcal{C}} := |0\rangle$, where P_i is while-free.

We claim **while** M_1 **do** S **done**; $p_{\mathcal{C}\otimes\mathcal{C}_3} := |0\rangle$ and

$$
\begin{aligned}
& g := |1\rangle\,; \\
& \textbf{while } \mathrm{Meas}[g] > 0 \textbf{ do} \\
& \quad \textbf{if } \mathrm{Meas}[g] = 1 \textbf{ then} \\
& \quad\quad \textbf{if } M_1 \textbf{ then } P_1; g := |2\rangle \\
& \quad\quad \textbf{else } g := |0\rangle \\
& \quad \textbf{else} \\
& \quad\quad \textbf{if } M_2 \textbf{ then } P_2 \\
& \quad\quad \textbf{else } g := |1\rangle \\
& p_{\mathcal{C}\otimes\mathcal{C}_3} := |0\rangle\,,
\end{aligned}
$$

are equivalent, whose encodings are $(m_{11}s)^* m_{12} c g^0$ and $g^1((g_1+g_2)(g_1(m_{11}p_1g^2+m_{12}g^0)+(g_0+g_2)(m_{21}p_2+m_{22}g^1)))^* g_0 c g^0$.

Similarly to the above case, utilizing inductive hypothesis, we have $sc = p_1(m_{21}p_2)^* m_{22} c$. Let $X = (g_1+g_2)(g_1(m_{11}p_1g^2+m_{12}g^0)+(g_0+g_2)(m_{21}p_2+m_{22}g^1)) = g_1(m_{11}p_1g^2+m_{12}g^0)+g_2(m_{21}p_2+m_{22}g^1)$. By denesting rule $g^1X^* = g^1(g_1(m_{11}p_1g^2+m_{12}g^0))^*$. Let $Y = g_1(m_{11}p_1g^2+m_{12}g^0)$, $Z = m_{11}p_1(m_{21}p_2)^* m_{22}$. So $g^2Y^* = g^2$ and

$$
\begin{aligned}
& g^1X^* = g^1Y^*(g_2(m_{21}p_2+m_{22}g^1)Y^*)^*, \\
& g^1Y^* = g^1(1+g_1(m_{11}p_1g^2+m_{12}g^0)(g_1(m_{11}p_1g^2+m_{12}g^0))^*) = g^1+m_{12}g^0+m_{11}p_1g^2.
\end{aligned}
$$

Hence $g^2(g_2m_{21}p_2Y^*)^* = g^2(g_2m_{21}p_2)^* = (m_{21}p_2)^*g^2$. By star-rewrite, there is

$$
\begin{aligned}
& g^2(g_2(m_{21}p_2 + m_{22}g^1)Y^*)^*g_0 \\
= {} & g^2(g_2m_{21}p_2Y^*)^*(g_2m_{22}g^1Y^*(g_2m_{21}p_2Y^*)^*)^*g_0 \\
= {} & (m_{21}p_2)^*g^2(g_2m_{22}(g^1 + m_{12}g^0 + m_{11}p_1g^2)(g_2m_{21}p_2Y^*)^*)^*g_0 \\
= {} & (m_{21}p_2)^*g^2(g_2m_{22}(g^1 + m_{12}g^0) + g_2m_{22}m_{11}p_1(m_{21}p_2)^*g^2)^*g_0 \\
= {} & (m_{21}p_2)^*g^2(g_2m_{22}m_{11}p_1(m_{21}p_2)^*g^2)^*(g_2m_{22}(g^1 + m_{12}g^0)(g_2m_{22}m_{11}p_1(m_{21}p_2)^*g^2)^*)^*g_0 \\
= {} & (m_{21}p_2)^*(m_{22}m_{11}p_1(m_{21}p_2)^*)^*g^2(g_2m_{22}(g^1 + m_{12}g^0))^*g_0
\end{aligned}
$$

Expand the star expression twice:

$$
\begin{aligned}
& g^2(g_2m_{22}(g^1 + m_{12}g^0))^*g_0 \\
= {} & g^2[1 + g_2m_{22}(g^1 + m_{12}g^0) + g_2m_{22}(g^1 + m_{12}g^0)g_2m_{22}(g^1 + m_{12}g^0)(g_2m_{22}(g^1 + m_{12}g^0))^*]g_0 \\
= {} & g^2(1 + g_2m_{22}(g^1 + m_{12}g^0))g_0 \\
= {} & m_{22}m_{12}g^0.
\end{aligned}
$$

By sliding,

$$
\begin{aligned}
g^2(g_2(m_{21}p_2 + m_{22}g^1)Y^*)^*g_0 &= (m_{21}p_2)^*(m_{22}m_{11}p_1(m_{21}p_2)^*)^*m_{22}m_{12}g^0 \\
&= (m_{21}p_2)^*m_{22}(m_{11}p_1(m_{21}p_2)^*m_{22})^*m_{12}g^0 \\
&= (m_{21}p_2)^*m_{22}Z^*m_{12}g^0.
\end{aligned}
$$

Combining them up gives

$$
\begin{aligned}
g^1 X^* g_0 &= (g^1 + m_{12}g^0 + m_{11}p_1 g^2)(g_2(m_{21}p_2 + m_{22}g^1)Y^*)^* g_0 \\
&= m_{12}g^0 + m_{11}p_1 g^2 (g_2(m_{21}p_2 + m_{22}g^1)Y^*)^* g_0 \\
&= (1 + m_{11}p_1(m_{21}p_2)^* m_{22} Z^*) m_{12} g^0 \\
&= (1 + ZZ^*) m_{12} g^0 \\
&= Z^* m_{12} g^0
\end{aligned}
$$

Hence we have

$$
\begin{aligned}
& g^1((g_1 + g_2)(g_1(m_{11}p_1 g^2 + m_{12}g^0) + (g_0 + g_2)(m_{21}p_2 + m_{22}g^1)))^* g_0 c g^0 \\
&= g^1 X^* g_0 c g^0 \\
&= Z^* m_{12} c g^0 \\
&= (m_{11}p_1(m_{21}p_2)^* m_{22})^* c m_{12} g^0 \\
&= (m_{11}p_1(m_{21}p_2)^* m_{22} c)^* c m_{12} g^0 \\
&= (m_{11} s c)^* c m_{12} g^0 \\
&= (m_{11} s)^* m_{12} c g^0.
\end{aligned}
$$

$\square$

8.7 Non-idempotent Kleene algebra with tests

As we stated before, NKA is not specifically designed for quantum programs: the measurements are treated as normal processes. Further characterization of measurements will grant finer algebraic structure. KAT introduces tests into KA, relying on the ability to simultaneously represent branching and predicates by Boolean algebra. However, for quantum programs, there is a gap between branching and predicates, which requires us to treat predicates and branching separately. We introduce effect algebra as a subalgebra of NKA to tackle quantum predicates in Section 8.7.1. As for branching, quantum measurements are abstracted as algebraic rules based on predicates. These lead to non-idempotent Kleene algebra with tests (NKAT) in Section 8.7.2. As an application, we show how propositional quantum Hoare logic is subsumed into algebraic rules of NKAT in Section 8.7.3 and Section 8.7.4.

8.7.1 Effect algebra

The notion of quantum predicates was defined in [203] as an operator $A \in \mathcal{PO}(\mathcal{H})$ satisfying $\|A\| \leq 1$, and its negation $\overline{A} = I_{\mathcal{H}} - A$. In the quantum foundations literature, quantum predicates are also called *effects*. Their algebraic properties have been extensively studied as effect algebras.

Definition 8.7.1 ([216])**.** *An effect algebra (EA) is a* 4*-tuple* $(\mathcal{L}, \oplus, 0, e)$*, where* $0, e \in \mathcal{L}$*, and* $\oplus : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$ *is a partial binary operation satisfying the following properties: for any* $a, b, c \in \mathcal{L}$*,*

1. *if* $a \oplus b$ *is defined then* $b \oplus a$ *is defined and* $a \oplus b = b \oplus a$*;*

2. *if* $a \oplus b$ *and* $(a \oplus b) \oplus c$ *are defined, then* $b \oplus c$ *and* $a \oplus (b \oplus c)$ *are defined and* $(a \oplus b) \oplus c = a \oplus (b \oplus c)$*;*

3. *if* $a \oplus e$ *is defined, then* $a = 0$*;*

4. *for every* $a \in \mathcal{L}$ *there exists a unique* $\overline{a} \in \mathcal{L}$ *such that* $a \oplus \overline{a} = e$*;*

5. *for every* $a \in \mathcal{L}$*,* $0 \oplus a = a$*.*

The fourth rule of the effect algebra defines a unary operator, the *negation* over $\mathcal{L}$, denoted by $\bar{a}$ for $a \in \mathcal{L}$.

An effect algebra is easily embedded in NKA by viewing $\oplus$ as a restricted $+$ of NKA. Then we need to identify the correspondence of predicates in the quantum path model.

Definition 8.7.2. *For a predicate* A*, a constant superoperator* $\mathcal{C}_A \in \mathcal{QC}(\mathcal{H})$ *for* $A \in \mathcal{PO}(\mathcal{H})$ *is defined by*

$$\mathcal{C}_A(\rho) = \mathrm{tr}[\rho]A. \tag{8.7.1.1}$$

We let $\mathcal{P}_{\mathrm{Pred}}(\mathcal{H}) = \{\langle \mathcal{C}_A \rangle^{\uparrow} : A \in \mathcal{PO}(\mathcal{H}), \|A\| \leq 1\}$ *be the subset of* $\mathcal{P}(\mathcal{H})$ *containing the lifted constant superoperator.*

A partial binary addition $\oplus$ *over* $\mathcal{P}_{\mathrm{Pred}}(\mathcal{H})$ *inherits from the addition in* $\mathcal{P}(\mathcal{H})$*, defined by:*

$$\langle \mathcal{C}_A \rangle^{\uparrow} \oplus \langle \mathcal{C}_B \rangle^{\uparrow} = \begin{cases} \langle \mathcal{C}_A \rangle^{\uparrow} + \langle \mathcal{C}_B \rangle^{\uparrow} & \langle \mathcal{C}_A \rangle^{\uparrow} + \langle \mathcal{C}_B \rangle^{\uparrow} \preceq \langle \mathcal{C}_{I_{\mathcal{H}}} \rangle^{\uparrow}, \\ \textit{undefined} & \textit{otherwise}. \end{cases}$$

Lemma 8.7.3. *$(\mathcal{P}_{\mathrm{Pred}}(\mathcal{H}), \oplus, \mathcal{O}_{\mathcal{H}}, \langle \mathcal{C}_{I_{\mathcal{H}}} \rangle^{\uparrow})$ forms an effect algebra. Specifically, the negation of it satisfies $\overline{\langle \mathcal{C}_A \rangle^{\uparrow}} = \langle \mathcal{C}_{\overline{A}} \rangle^{\uparrow}$.*

Proof of Lemma 8.7.3.

1. If $\langle \mathcal{C}_A \rangle^{\uparrow} \oplus \langle \mathcal{C}_B \rangle^{\uparrow}$ is defined, then $\langle \mathcal{C}_A \rangle^{\uparrow} + \langle \mathcal{C}_B \rangle^{\uparrow} \preceq \langle \mathcal{C}_{I_{\mathcal{H}}} \rangle^{\uparrow}$. Commutativity of addition makes $\langle \mathcal{C}_B \rangle^{\uparrow} + \langle \mathcal{C}_A \rangle^{\uparrow} \preceq \langle \mathcal{C}_{I_{\mathcal{H}}} \rangle^{\uparrow}$ and leads to $\langle \mathcal{C}_B \rangle^{\uparrow} \oplus \langle \mathcal{C}_A \rangle^{\uparrow} = \langle \mathcal{C}_B \rangle^{\uparrow} + \langle \mathcal{C}_A \rangle^{\uparrow}$.

2. If $\langle \mathcal{C}_A \rangle^{\uparrow} \oplus \langle \mathcal{C}_B \rangle^{\uparrow}$ and $(\langle \mathcal{C}_A \rangle^{\uparrow} \oplus \langle \mathcal{C}_B \rangle^{\uparrow}) \oplus \langle \mathcal{C}_C \rangle^{\uparrow}$ are defined, then $\langle \mathcal{C}_A \rangle^{\uparrow} + \langle \mathcal{C}_B \rangle^{\uparrow} \preceq \langle \mathcal{C}_{I_{\mathcal{H}}} \rangle^{\uparrow}$ and $(\langle \mathcal{C}_A \rangle^{\uparrow} + \langle \mathcal{C}_B \rangle^{\uparrow}) + \langle \mathcal{C}_C \rangle^{\uparrow} \preceq \langle \mathcal{C}_{I_{\mathcal{H}}} \rangle^{\uparrow}$. Hence $\langle \mathcal{C}_B \rangle^{\uparrow} + \langle \mathcal{C}_C \rangle^{\uparrow} \preceq \langle \mathcal{C}_{I_{\mathcal{H}}} \rangle^{\uparrow}$ and $\langle \mathcal{C}_A \rangle^{\uparrow} + (\langle \mathcal{C}_B \rangle^{\uparrow} + \langle \mathcal{C}_C \rangle^{\uparrow}) \preceq \langle \mathcal{C}_{I_{\mathcal{H}}} \rangle^{\uparrow}$. By definition, $\langle \mathcal{C}_B \rangle^{\uparrow} + \langle \mathcal{C}_C \rangle^{\uparrow}$ and $\langle \mathcal{C}_A \rangle^{\uparrow} + (\langle \mathcal{C}_B \rangle^{\uparrow} + \langle \mathcal{C}_C \rangle^{\uparrow})$ are defined, and $\langle \mathcal{C}_A \rangle^{\uparrow} + (\langle \mathcal{C}_B \rangle^{\uparrow} + \langle \mathcal{C}_C \rangle^{\uparrow}) = (\langle \mathcal{C}_A \rangle^{\uparrow} + \langle \mathcal{C}_B \rangle^{\uparrow}) + \langle \mathcal{C}_C \rangle^{\uparrow}$.

3. If $\langle \mathcal{C}_A \rangle^{\uparrow} \oplus \langle \mathcal{C}_{I_{\mathcal{H}}} \rangle^{\uparrow}$ is defined in $\mathcal{P}_{\mathrm{Pred}}(\mathcal{H})$, we assume $\langle \mathcal{C}_A \rangle^{\uparrow} + \langle \mathcal{C}_{I_{\mathcal{H}}} \rangle^{\uparrow} = \langle \mathcal{C}_B \rangle^{\uparrow}$. Apply the quantum actions on $[|0\rangle \langle 0|]$, we have $[A + I_{\mathcal{H}}] = [B]$. Meanwhile, $\|A\|, \|B\| \leq 1$. This forces $A = 0$ so $\langle \mathcal{C}_A \rangle^{\uparrow} = \langle \mathcal{C}_{O_{\mathcal{H}}} \rangle^{\uparrow} = \mathcal{O}_{\mathcal{H}}$.

4. For $\langle \mathcal{C}_A \rangle^{\uparrow} \in \mathcal{P}_{\mathrm{Pred}}(\mathcal{H})$, there is $\langle \mathcal{C}_A \rangle^{\uparrow} + \langle \mathcal{C}_{\overline{A}} \rangle^{\uparrow} = \langle \mathcal{C}_{I_{\mathcal{H}}} \rangle^{\uparrow}$, hence $\langle \mathcal{C}_A \rangle^{\uparrow} \oplus \langle \mathcal{C}_{\overline{A}} \rangle^{\uparrow} = \langle \mathcal{C}_{I_{\mathcal{H}}} \rangle^{\uparrow}$. Meanwhile, if $\langle \mathcal{C}_A \rangle^{\uparrow} \oplus \langle \mathcal{C}_B \rangle^{\uparrow} = \langle \mathcal{C}_{I_{\mathcal{H}}} \rangle^{\uparrow}$, we apply these quantum actions on $[|0\rangle \langle 0|]$, resulting in $[A + B] = [I_{\mathcal{H}}]$. Hence $B = I - A = \overline{A}$. That is, $\overline{\langle \mathcal{C}_A \rangle^{\uparrow}} = \langle \mathcal{C}_{\overline{A}} \rangle^{\uparrow}$ is the unique negation of $\langle \mathcal{C}_A \rangle^{\uparrow}$ in $\mathcal{P}_{\mathrm{Pred}}(\mathcal{H})$.

5. For $\langle \mathcal{C}_A \rangle^{\uparrow} \in \mathcal{P}_{\mathrm{Pred}}(\mathcal{H})$, $\mathcal{O}_{\mathcal{H}} + \langle \mathcal{C}_A \rangle^{\uparrow} = \langle \mathcal{C}_A \rangle^{\uparrow} \preceq \langle \mathcal{C}_{I_{\mathcal{H}}} \rangle^{\uparrow}$, whose left hand side then equals to $\mathcal{O}_{\mathcal{H}} \oplus \langle \mathcal{C}_A \rangle^{\uparrow}$ by definition.

□

8.7.2 Non-idempotent Kleene algebras with tests

We can characterize quantum measurements with the help of predicates, for which we propose *partitions* algebraically.

Definition 8.7.4. *An* NKAT *is a many-sorted algebra* $(\mathcal{K}, \mathcal{L}, \mathcal{N}, +, \cdot, *, \leq, 0, 1, e)$ *such that*

1. $(\mathcal{K}, +, \cdot, *, \leq, 0, 1)$ *is an* NKA*;*

2. $\mathcal{L}$ *is a subset of* $\mathcal{K}$*, and* $(\mathcal{L}, \oplus, 0, e)$ *is an effect algebra, where* $\oplus$ *is the restriction of* $+$ *w.r.t. top element* e *and partial order* $\leq$*; that is, for any* $a, b \in \mathcal{L}$

$$a \oplus b = \begin{cases} a + b & a + b \leq e, \\ \textit{undefined} & \textit{otherwise}; \end{cases} \tag{8.7.2.1}$$

3. $\mathcal{N}$ *is a set of tuples* $(m_i)_{i \in I}$*, where* I *are finite index sets and* $m_i \in \mathcal{K}$*, satisfying:*

   (a) *each entry in the tuples satisfies* $m_i \mathcal{L} \subseteq \mathcal{L}$*; that is, for* $a \in \mathcal{L}$*,* $m_i a \in \mathcal{L}$*.*

   (b) *for each tuple,* $\sum_{i \in I} m_i e = e$*.*

   The tuples in $\mathcal{N}$ *are called* partitions*.*

We use $\mathcal{L}$ to characterize quantum predicates, and $\mathcal{N}$ to characterize branching in quantum programs. For a quantum measurement $\{M_i\}_{i \in I}$, its dual superoperators transform quantum predicates to quantum predicates: $\mathcal{E}^{\dagger}_{M_i}(A) = M_i^{\dagger} A M_i$. This is captured by $m_i \mathcal{L} \subseteq \mathcal{L}$. Besides, general quantum measurements satisfies $\sum_{i \in I} M_i^{\dagger} M_i = I$, which is captured by $\sum_{i \in I} m_i e = e$, since e represents predicate $I_{\mathcal{H}}$. [2]

[2] Our characterization of measurements matches positive-operator-valued measurements (POVM), the most gen-

Definition 8.7.5. *The set of quantum measurements lifted as quantum path actions in the dual sense is* $\mathcal{P}_{\text{Meas}}(\mathcal{H}) = \left\{ \left(\langle \mathcal{M}_i^\dagger \rangle^\uparrow \right)_{i \in I} \ : \ \mathcal{M}_i(\rho) = M_i \rho M_i^\dagger, \sum_{i \in I} M_i^\dagger M_i = I_{\mathcal{H}} \right\}$.

Then we augment the quantum path model in the NKAT framework, supporting quantum predicates ($\mathcal{P}_{\text{Pred}}(\mathcal{H})$) and quantum measurements ($\mathcal{P}_{\text{Meas}}(\mathcal{H})$).

Theorem 8.7.6. *The NKAT axioms are sound for the algebra* $(\mathcal{P}(\mathcal{H}), \mathcal{P}_{\text{Pred}}(\mathcal{H}), \mathcal{P}_{\text{Meas}}(\mathcal{H}), +, \diamond, *, \preceq, \mathcal{O}_{\mathcal{H}}, \mathcal{I}_{\mathcal{H}}, \langle \mathcal{C}_{I_{\mathcal{H}}} \rangle^\uparrow)$.

Proof of Theorem 8.7.6. Notice that the NKA axioms are symmetric for operands of $\cdot$. That is, if we define $a \star b = b \cdot a$, any axiom substituting $\star$ for $\cdot$ has a corresponding axiom. Hence if $(\mathcal{K}, +, \cdot, *, 0, 1)$ forms an NKA, $(\mathcal{K}, +, \star, *, 0, 1)$ also forms an NKA. Hence Theorem 8.3.8 has verified (1) in Definition 8.7.4.

Meanwhile, Lemma 8.7.3 has verified (2) in Definition 8.7.4. We only need to verify (3) here.

- By definition, $\langle \mathcal{M}_i^\dagger \rangle^\uparrow$ are elements of $\mathcal{P}(\mathcal{H})$.
- Note $\langle \mathcal{M}_i^\dagger \rangle^\uparrow \diamond \langle \mathcal{C}_A \rangle^\uparrow (\sum_j [\rho_j]) = \sum_j [\text{tr}(\rho_j) \mathcal{M}_i^\dagger A \mathcal{M}_i] = \langle \mathcal{C}_{\mathcal{M}_i^\dagger A \mathcal{M}_i} \rangle^\uparrow (\sum_j [\rho_j])$. Hence $\langle \mathcal{M}_i^\dagger \rangle^\uparrow \diamond \langle \mathcal{C}_A \rangle^\uparrow = \langle \mathcal{C}_{\mathcal{M}_i^\dagger A \mathcal{M}_i} \rangle^\uparrow$ and it is in $\mathcal{P}_{\text{Pred}}(\mathcal{H})$.

eral quantum measurements. We can further classify structures inside $\mathcal{N}$ to depict specific classes of quantum measurements. For example, projection-valued measurements (PVM) can be modeled as tuples $(m_i)_{i \in I}$ where $m_i m_j = m_i$ if $i = j$, otherwise $m_i m_j = 0$.

Furthermore, a set of projective and pair-wise commutative measurement superoperators, defined by $C(\mathcal{H}) = \{\mathcal{E} \in \mathcal{QC}(\mathcal{H}) : \mathcal{E}(\rho) = D \rho D^\dagger, D \text{ is diagonal}, D^2 = D\}$, represents the measurement superoperators in probabilistic programs. A Boolean algebra can be observed from it. It would be an interesting future direction to investigate the algebraic relation between NKAT and this Boolean algebra.

- Similarly, we have

$$\begin{aligned}
&\left(\sum_{i\in I}\langle\mathcal{M}_i^\dagger\rangle^\uparrow \diamond \langle\mathcal{C}_{I_\mathcal{H}}\rangle^\uparrow\right)\left(\sum_j[\rho_j]\right)\\
=\ &\sum_j\left[\mathrm{tr}(\rho_j)\sum_{i\in I}\mathcal{M}_i^\dagger\mathcal{M}_i\right]\\
=\ &\sum_j[\mathrm{tr}(\rho_j)I_\mathcal{H}] = \langle\mathcal{C}_{I_\mathcal{H}}\rangle^\uparrow\left(\sum_j[\rho_j]\right).
\end{aligned}$$

This gives $\left(\sum_{i\in I}\langle\mathcal{M}_i^\dagger\rangle^\uparrow \diamond \langle\mathcal{C}_{I_\mathcal{H}}\rangle^\uparrow\right) = \langle\mathcal{C}_{I_\mathcal{H}}\rangle^\uparrow$.

□

Note we have substituted the right composition operation $\diamond$ for the left composition operation ; in $\mathcal{P}(\mathcal{H})$. This is mainly because our interpretation now uses dual superoperators. The verification of each axiom is standard.

Several useful rules are derivable in NKAT.

Lemma 8.7.7. *The following formulae are derivable in NKAT. Here I is a finite index set, a, b, a_i are elements of the effect subalgebra, $(m_i)_{i\in I}$ is a partition.*

1. $0 \leq a \leq e$;
2. $a + \overline{a} = e$;
3. $\overline{\overline{a}} = a$;
4. *(negation-reverse):* $a \leq b \to \overline{b} \leq \overline{a}$;
5. *(partition-transform):* $\overline{\sum_{i\in I} m_i a_i} = \sum_{i\in I} m_i \overline{a_i}$.

Proof of Lemma 8.7.7.

- ($0 \leq a \leq e$): Notice that $0 \oplus a = a$ is defined by the definition of effect algebra. There is $0 \leq 0 + a = a \leq e$.

- ($a + \overline{a} = e$): Because $a \oplus \overline{a} = e$ is defined, we have $e = a \oplus \overline{a} = a + \overline{a}$.

- ($\overline{\overline{a}} = a$): Notice that there exists a unique $\overline{a} \in \mathcal{L}$ satisfying $a \oplus \overline{a} = e$. Then there exists a unique $\overline{\overline{a}}$ satisfying $\overline{\overline{a}} \oplus \overline{a} = e$. Therefore $a = \overline{\overline{a}}$.

- (negation-reverse): Because $a \leq b$, $0 \leq a + \overline{b} \leq b + \overline{b} = e$. Hence $a \oplus \overline{b} \in \mathcal{L}$. Let $c = \overline{a \oplus \overline{b}} \in \mathcal{L}$, there is $0 \leq c$. So $a \oplus \overline{b} \oplus c = e = a \oplus \overline{a}$. Thence $\overline{a} = \overline{b} \oplus c = \overline{b} + c$, and $\overline{a} \leq \overline{b}$.

- (partition-transform): By $0 \leq a_i \leq e$, monotone properties and $m_i a_i \in \mathcal{L}$, $\sum_{i \in I} m_i a_i \leq \sum_{i \in I} m_i e = e$. So $\bigoplus_{i \in I} m_i a_i = \sum_{i \in I} m_i a_i \in \mathcal{L}$ by the definition of $\oplus$. Similarly $\bigoplus_{i \in I} m_i \overline{a_i} = \sum_{i \in I} m_i \overline{a_i} \in \mathcal{L}$. Adding them together, $e = \sum_{i \in I} m_i e = \sum_{i \in I} m_i (a_i + \overline{a_i}) = \sum_{i \in I} m_i a_i + \sum_{i \in I} m_i \overline{a_i}$. Hence $\overline{\sum_{i \in I} m_i a_i} = \sum_{i \in I} m_i \overline{a_i}$.

□

8.7.3 Encoding of quantum Hoare triples

A natural usage of classical predicates is reasoning via Hoare triples. With an algebraic representation of quantum predicates and programs, we can encode quantum Hoare triples as algebraic formulae. A quantum Hoare triple is a judgment of the form $\{A\}P\{B\}$ where A, B are quantum predicates and P is a quantum program. It refers to partial correctness [234], denoted

by $\models_{par} \{A\}P\{B\}$, if for all input $\rho \in \mathcal{D}(\mathcal{H})$ there is

$$\mathrm{tr}(A\rho) \leq \mathrm{tr}(B \llbracket P \rrbracket (\rho)) + \mathrm{tr}(\rho) - \mathrm{tr}\left(\llbracket P \rrbracket (\rho)\right). \tag{8.7.3.1}$$

Then partial correctness $\models_{par} \{A\}P\{B\}$ can be encoded as an inequality $p\bar{b} \leq \bar{a}$, where p is the encoding of program P, and effect algebra elements a, b are the encoding of constant superoperators $\mathcal{C}_A$ and $\mathcal{C}_B$. This encoding can be interpreted by a dual interpretation $\mathcal{Q}^{\dagger}_{\mathrm{int}}$. [3] By setting any non-zero input for $\mathcal{Q}^{\dagger}_{\mathrm{int}}(p\bar{b}) \preceq \mathcal{Q}^{\dagger}_{\mathrm{int}}(\bar{a})$ and Lemma 8.3.10.(ii), it turns to $\llbracket P \rrbracket^{\dagger} (I - B) \sqsubseteq I - A$, which is equivalent to $\models_{par} \{A\}P\{B\}$.

8.7.4 Propositional quantum Hoare logic

An important feature of KAT is that KAT subsumes the deductive system of propositional Hoare logic, which contains the rules directly related to the control flow of classical while programs but not the rule for assignments [180]. As a counterpart, quantum Hoare logic is an important tool in the verification and analysis of quantum programs. A sound and (relatively) complete proof system for partial correctness of quantum **while** programs presented in Figure 8.5 is discussed in [202]. We aim to subsume in NKAT a fragment of quantum Hoare logic, called *propositional* quantum Hoare logic.

Due to the no-cloning of quantum information, the role of assignment is played by initialization and unitary transformation together in quantum programming. In quantum Hoare logic, the rule (Ax.In) and (Ax.UT) for them include atomic transformations, which cannot be captured

[3] A *dual interpretation* $\mathcal{Q}^{\dagger}_{\mathrm{int}}$ is defined similar to $\mathcal{Q}_{\mathrm{int}}$ except for $\mathcal{Q}^{\dagger}_{\mathrm{int}}(e \cdot f) = \mathcal{Q}^{\dagger}_{\mathrm{int}}(e) \diamond \mathcal{Q}^{\dagger}_{\mathrm{int}}(f)$ and $\mathcal{Q}^{\dagger}_{\mathrm{int}}(a) = \langle \mathrm{eval}(a)^{\dagger} \rangle^{\uparrow}$. It describes the dual superoperators lifted to $\mathcal{P}(\mathcal{H})$. Properties of $\mathcal{Q}_{\mathrm{int}}$ like Theorem 8.4.2, Corollary 8.4.3 hold for the dual interpretation similarly. Analogous of Theorem 8.4.5, $\mathcal{Q}^{\dagger}_{\mathrm{int}}(\mathrm{Enc}(P)) = \langle \llbracket P \rrbracket^{\dagger} \rangle^{\uparrow}$, holds as well.

$$\text{(Ax.UT)}\quad \{U^\dagger A U\}\ \overline{q} := U[\overline{q}]\ \{A\} \qquad \text{(Ax.In)}\quad \left\{\sum_i |i\rangle_q \langle 0| A |0\rangle_q \langle i|\right\} q := |0\rangle\ \{A\}$$

$$\text{(Ax.Sk)}\quad \{A\}\ \mathbf{skip}\ \{A\} \qquad \text{(R.OR)}\quad \frac{A \sqsubseteq A' \quad \{A'\}P\{B'\} \quad B' \sqsubseteq B}{\{A\}P\{B\}}$$

$$\text{(Ax.Ab)}\quad \{I_{\mathcal{H}}\}\ \mathbf{abort}\ \{O_{\mathcal{H}}\} \qquad \text{(R.IF)}\quad \frac{\{A_i\}P_i\{B\} \text{ for all } i}{\{\sum_i \mathcal{M}_i^\dagger(A_i)\}\mathbf{case}\ M \xrightarrow{i} P_i\ \mathbf{end}\{B\}}$$

$$\text{(R.SC)}\quad \frac{\{A\}P_1\{B\} \quad \{B\}P_2\{C\}}{\{A\}P_1; P_2\{C\}} \qquad \text{(R.LP)}\quad \frac{\{B\}P\{C\} \quad C = \mathcal{M}_0^\dagger(A) + \mathcal{M}_1^\dagger(B)}{\{C\}\mathbf{while}\ M = 1\ \mathbf{do}\ P\ \mathbf{done}\{A\}}$$

Figure 8.5: A proof system for partial correctness of quantum programs. Propositional quantum Hoare logic includes the rules marked red in this figure (the lower six rules).

by algebraic methods. As such, propositional quantum Hoare logic will treat these rules as atomic propositions and work with the following program syntax

$$\begin{aligned} P \ ::= \ & p \mid \mathbf{skip} \mid \mathbf{abort} \mid P_1; P_2 \mid \\ & \mathbf{case}\ M[\overline{q}] \xrightarrow{i} P_i\ \mathbf{end} \mid \mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ P_1\ \mathbf{done}. \end{aligned}$$

Therefore, the deductive system of propositional quantum Hoare logic consists of the rules marked red in Figure 8.5. Its relative completeness and soundness can be proved similarly to the original quantum Hoare logic [202] as a routine exercise.

By the discussions in Section 8.7.3, the partial correctness of quantum Hoare triples can be encoded in NKAT. For a quantum measurement $\{M_i\}_{i\in I}$ we have an additional normalization

rule $\sum_i M_i^\dagger M_i = I$, which is encoded as $\sum_i m_i e = e$. Then the encoding of these rules is

$$\begin{cases} \text{(Ax.Sk)}: & 1\overline{a} \leq \overline{a}, \\ \text{(Ax.Ab)}: & 0\overline{0} \leq \overline{1}, \\ \text{(R.OR)}: & a \leq a' \wedge p\overline{b'} \leq \overline{a'} \wedge b' \leq b \to p\overline{b} \leq \overline{a}, \\ \text{(R.IF)}: & \left(\bigwedge_{i\in I} p_i\overline{b} \leq \overline{a_i}\right) \to (\sum_{i\in I} m_i p_i)\overline{b} \leq \overline{\sum_i m_i a_i}, \\ \text{(R.SC)}: & p_1\overline{b} \leq \overline{a} \wedge p_2\overline{c} \leq \overline{b} \to p_1 p_2 \overline{c} \leq \overline{a}, \\ \text{(R.LP)}: & p\overline{m_0 a + m_1 b} \leq \overline{b} \to (m_1 p)^* m_0 \overline{a} \leq \overline{m_0 a + m_1 b}. \end{cases}$$

Here I is a finite index set, $p, p_i \in \mathcal{K}$, elements $a, b, c, a', b', a_i \in \mathcal{L}$, and $(m_i)_{i\in I}$ are partitions.

Theorem 8.7.8. *With partitions $(m_i)_{i\in I}$, the formulae above are derivable in NKAT.*

Proof.

1. (Ax.Sk): $1\overline{a} = \overline{a}$.

2. (Ax.Ab): $0\overline{0} = 0 \leq \overline{1}$ by positivity.

3. (R.OR): By negation-reverse, we have $\overline{a'} \leq \overline{a}$ and $\overline{b} \leq \overline{b'}$. So $p\overline{b} \leq p\overline{b'} \leq \overline{a'} \leq \overline{a}$.

4. (R.IF): Applying partition-transform, $(\sum_{i\in I} m_i p_i)\overline{b} = \sum_{i\in I} m_i p_i \overline{b} \leq \sum_{i\in I} m_i \overline{a_i} = \overline{\sum_i m_i a_i}$.

5. (R.SC): $p_1(p_2\overline{c}) \leq p_1\overline{b} \leq \overline{a}$.

6. (R.LP): By partition-transform, $\overline{m_0 a + m_1 b} = m_0\overline{a} + m_1\overline{b}$. With $p\overline{m_0 a + m_1 b} \leq \overline{b}$, we

have

$$m_0\overline{a} + m_1 p\overline{m_0 a + m_1 b} \leq m_0\overline{a} + m_1\overline{b} = \overline{m_0 a + m_1 b}.$$

Then $(m_1 p)^* m_0\overline{a} \leq \overline{m_0 a + m_1 b}$ is concluded by applying $q + pr \leq r \rightarrow p^* q \leq r$.

$\square$

It is clear that the NKAT subsumes the encoding of propositional quantum Hoare logic.

Part IV

Conclusion

Conclusion

In this dissertation, we have studied software tools for quantum computing, from building software stacks to leveraging formal methods to ensure the correctness of the software. We established the foundation of Hamiltonian-oriented programming software and then illustrated an example of theoretical tool design for the control software of HOQC. We also justified the feasibility of deductive verification for ensuring the correctness of COQC quantum algorithm implementation and built a theory for leveraging algebraic expression derivation for quantum program equivalences.

From the discussions in the dissertation, we had an overarching understanding of the development of quantum software. Ideally, it depends on three aspects:

1. Build an end-to-end software stack connecting applications to quantum hardware.

2. Explore practical applications using the software stack.

3. Ensuring the correctness of the software stack.

The developments of HOQC and COQC software are still limited in this picture. There are many directions to be further explored in the future.

The future directions after this dissertation include filling in missing pieces for high-assurance software stacks for HOQC and COQC. Several possible directions include

- Build software to fill in the gaps in the software stack for HOQC.
- Optimize the software stack for specific applications and specific hardware.
- Develop formal methods for HOQC software stacks.
- Develop methods to verify the hardware for both HOQC and COQC.
- Investigate proper high-level abstractions for circuit-oriented programming.

Upon finishing this dissertation, we are specifically interested in exploring practical applications with our software stack based on SimuQ. Benefiting from the low overheads of HOQC and large-scale analog quantum simulators, we have observed evidence of practical usage of extant quantum computers. We hope that we can achieve real-world quantum advantages shortly with the help of the high-assurance software tools we built in this dissertation.

Bibliography

[1] RP Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, 1982. doi: https://doi.org/10.1007/BF02650179.

[2] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997. ISSN 0097-5397. doi: 10.1137/S0097539795293172. URL `https://doi.org/10.1137/S0097539795293172`.

[3] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. A verified optimizer for quantum circuits. *Proc. ACM Program. Lang.*, 5(POPL), jan 2021. doi: 10.1145/3434318. URL `https://doi.org/10.1145/3434318`.

[4] Junyi Liu, Bohua Zhan, Shuling Wang, Shenggang Ying, Tao Liu, Yangjia Li, Mingsheng Ying, and Naijun Zhan. Formal verification of quantum algorithms using quantum hoare logic. In *International conference on computer aided verification*, pages 187–207. Springer, 2019.

[5] Christophe Chareton, Sébastien Bardin, François Bobot, Valentin Perrelle, and Benoît Valiron. An automated deductive verification framework for circuit-building quantum programs. *Programming Languages and Systems: 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 –April 1, 2021, Proceedings*, 12648:148–177, 03 2021. doi: 10.1007/978-3-030-72019-3{_}6. URL `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7984546/`.

[6] Dexter Kozen. A completeness theorem for kleene algebras and the algebra of regular events. Technical report, Cornell University, 1990.

[7] Allegra Angus and Dexter Kozen. Kleene algebra with tests and program schematology. Technical Report TR2001-1844, Computer Science Department, Cornell University, July 2001.

[8] Dexter Kozen. Kleene algebra with tests. *ACM Trans. Programming Languages and Systems (TOPLAS)*, 19(3):427–443, May 1997. doi: 10.1145/256167.256195.

[9] Dexter Kozen. On hoare logic and kleene algebra with tests. *ACM Trans. Comput. Logic*, 1(1):60–76, July 2000. ISSN 1529-3785. doi: 10.1145/343369.343378. URL `https://doi.org/10.1145/343369.343378`.

[10] Yuxiang Peng, Jacob Young, Pengyu Liu, and Xiaodi Wu. Simuq: A framework for programming quantum hamiltonian simulation with analog compilation (extended version). *arXiv preprint arXiv:2303.02775*, 2023. doi: https://doi.org/10.48550/arXiv.2303.02775.

[11] Jiaqi Leng, Yuxiang Peng, Yi-Ling Qiao, Ming Lin, and Xiaodi Wu. Differentiable analog quantum computing for optimization and control. *Advances in Neural Information Processing Systems*, 35:4707–4721, 2022.

[12] John Backus. *The History of Fortran I, II, and III*, page 25–74. Association for Computing Machinery, New York, NY, USA, 1978. ISBN 0127450408. URL `https://doi.org/10.1145/800025.1198345`.

[13] Kristen Nygaard and Ole-Johan Dahl. *The Development of the SIMULA Languages*, page 439–480. Association for Computing Machinery, New York, NY, USA, 1978. ISBN 0127450408. URL `https://doi.org/10.1145/800025.1198392`.

[14] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[15] Andrew M Childs. Lecture notes on quantum algorithms. *Lecture notes at University of Maryland*, 2017.

[16] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: A scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, page 333–342, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450320146. doi: https://doi.org/10.1145/2491956.2462177. URL `https://doi.org/10.1145/2491956.2462177`.

[17] Ali J Abhari, Arvin Faruque, Mohammad J Dousti, Lukas Svec, Oana Catu, Amlan Chakrabati, Chen-Fu Chiang, Seth Vanderwilt, John Black, and Fred Chong. Scaffold: Quantum programming language. Technical report, Princeton Univ NJ Dept of Computer Science, 2012.

[18] Jennifer Paykin, Robert Rand, and Steve Zdancewic. QWIRE: A core language for quantum circuits. *SIGPLAN Not.*, 52(1):846–858, January 2017. ISSN 0362-1340. doi: 10.1145/3093333.3009894. URL `https://doi.org/10.1145/3093333.3009894`.

[19] Yudong Cao, Jonathan Romero, Jonathan P Olson, Matthias Degroote, Peter D Johnson, Mária Kieferová, Ian D Kivlichan, Tim Menke, Borja Peropadre, Nicolas PD Sawaya, et al. Quantum chemistry in the age of quantum computing. *Chemical reviews*, 119(19): 10856–10915, 2019. doi: https://doi.org/10.1021/acs.chemrev.8b00803.

[20] Benjamin Nachman, Davide Provasoli, Wibe A De Jong, and Christian W Bauer. Quantum algorithm for high energy physics simulations. *Physical review letters*, 126(6):062001, 2021. doi: https://doi.org/10.1103/PhysRevLett.126.062001.

[21] Walter Hofstetter and Tao Qin. Quantum simulation of strongly correlated condensed matter systems. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 51(8): 082001, 2018. doi: 10.1088/1361-6455/aaa31b.

[22] WMC Foulkes, Lubos Mitas, RJ Needs, and Guna Rajagopal. Quantum monte carlo simulations of solids. *Reviews of Modern Physics*, 73(1):33, 2001. doi: https://doi.org/10.1103/RevModPhys.73.33.

[23] Ulrich Schollwöck. The density-matrix renormalization group. *Reviews of modern physics*, 77(1):259, 2005. doi: https://doi.org/10.1103/RevModPhys.77.259.

[24] Ulrich Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of physics*, 326(1):96–192, 2011. doi: https://doi.org/10.1016/j.aop.2010.09.012.

[25] Jeremy L. O'brien, Akira Furusawa, and Jelena Vučković. Photonic quantum technologies. *Nature Photonics*, 3(12):687–695, 2009. doi: https://doi.org/10.1038/nphoton.2009.229.

[26] Göran Wendin. Quantum information processing with superconducting circuits: a review. *Reports on Progress in Physics*, 80(10):106001, 2017. doi: 10.1088/1361-6633/aa7e1a.

[27] Christoph Kloeffel and Daniel Loss. Prospects for spin-based quantum computing in quantum dots. *Annu. Rev. Condens. Matter Phys.*, 4(1):51–81, 2013. doi: https://doi.org/10.1146/annurev-conmatphys-030212-184248.

[28] Mark Saffman. Quantum computing with atomic qubits and rydberg interactions: progress and challenges. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 49(20): 202001, 2016. doi: 10.1088/0953-4075/49/20/202001.

[29] Colin D Bruzewicz, John Chiaverini, Robert McConnell, and Jeremy M Sage. Trapped-ion quantum computing: Progress and challenges. *Applied Physics Reviews*, 6(2), 2019. doi: https://doi.org/10.1063/1.5088164.

[30] A Yu Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191, 1997. doi: 10.1070/RM1997v052n06ABEH002155.

[31] Andrew Cross. The ibm q experience and qiskit open-source quantum computing software. In *APS March meeting abstracts*, volume 2018, pages L58–003, 2018.

[32] Shantanu Debnath, Norbert M Linke, Caroline Figgatt, Kevin A Landsman, Kevin Wright, and Christopher Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536(7614):63–66, 2016. doi: https://doi.org/10.1038/nature18648.

[33] Daniel Gottesman. An introduction to quantum error correction and fault-tolerant quantum computation. In *Quantum information science and its contributions to mathematics, Proceedings of Symposia in Applied Mathematics*, volume 68, pages 13–58, 2010. doi: https://doi.org/10.48550/arXiv.0904.2557.

[34] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018. doi: https://doi.org/10.22331/q-2018-08-06-79.

[35] Erez Zohar, J Ignacio Cirac, and Benni Reznik. Quantum simulations of lattice gauge theories using ultracold atoms in optical lattices. *Reports on Progress in Physics*, 79(1): 014401, 2015. doi: 10.1088/0034-4885/79/1/014401.

[36] Alexey Vyacheslavovich Gorshkov, M Hermele, V Gurarie, C Xu, Paul S Julienne, J Ye, Peter Zoller, Eugene Demler, Mikhail D Lukin, and AM Rey. Two-orbital su (n) magnetism with ultracold alkaline-earth atoms. *Nature physics*, 6(4):289–295, 2010. doi: https://doi.org/10.1038/nphys1535.

[37] Sepehr Ebadi, Tout T Wang, Harry Levine, Alexander Keesling, Giulia Semeghini, Ahmed Omran, Dolev Bluvstein, Rhine Samajdar, Hannes Pichler, Wen Wei Ho, et al. Quantum phases of matter on a 256-atom programmable quantum simulator. *Nature*, 595(7866): 227–232, 2021. doi: https://doi.org/10.1038/s41586-021-03582-4.

[38] Bing Yang, Hui Sun, Robert Ott, Han-Yi Wang, Torsten V Zache, Jad C Halimeh, Zhen-Sheng Yuan, Philipp Hauke, and Jian-Wei Pan. Observation of gauge invariance in a 71-site bose–hubbard quantum simulator. *Nature*, 587(7834):392–396, 2020. doi: https://doi.org/10.1038/s41586-020-2910-8.

[39] Yunong Shi, Pranav Gokhale, Prakash Murali, Jonathan M. Baker, Casey Duckering, Yongshan Ding, Natalie C. Brown, Christopher Chamberland, Ali Javadi-Abhari, Andrew W. Cross, David I. Schuster, Kenneth R. Brown, Margaret Martonosi, and Frederic T. Chong. Resource-efficient quantum computing by breaking abstractions. *Proceedings of the IEEE*, 108(8):1353–1370, 2020. doi: 10.1109/JPROC.2020.2994765.

[40] Andrew Cross, Ali Javadi-Abhari, Thomas Alexander, Niel De Beaudrap, Lev S. Bishop, Steven Heidel, Colm A. Ryan, Prasahnt Sivarajah, John Smolin, Jay M. Gambetta, and Blake R. Johnson. Openqasm 3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing*, 3(3), sep 2022. ISSN 2643-6809. doi: 10.1145/3505636. URL `https://doi.org/10.1145/3505636`.

[41] QuEra. Bloqade: a julia package for quantum computation and quantum dynamics based on neutral-atom architectures, 2022. URL `https://queracomputing.github.io/Bloqade.jl/dev/`.

[42] Henrique Silvério, Sebastián Grijalva, Constantin Dalyac, Lucas Leclerc, Peter J. Karalekas, Nathan Shammah, Mourad Beji, Louis-Paul Henry, and Loïc Henriet. Pulser: An open-source package for the design of pulse sequences in programmable neutral-atom arrays. *Quantum*, 6:629, January 2022. ISSN 2521-327X. doi: 10.22331/q-2022-01-24-629. URL `https://doi.org/10.22331/q-2022-01-24-629`.

[43] J.R. Johansson, P.D. Nation, and Franco Nori. Qutip: An open-source python framework for the dynamics of open quantum systems. *Computer Physics Communications*, 183(8):1760–1772, 2012. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2012.

02.021. URL https://www.sciencedirect.com/science/article/pii/S0010465512000835.

[44] Gushu Li, Anbang Wu, Yunong Shi, Ali Javadi-Abhari, Yufei Ding, and Yuan Xie. Paulihedral: A generalized block-wise compiler optimization framework for quantum simulation kernels. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, page 554–569, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392051. doi: 10.1145/3503222.3507715. URL https://doi.org/10.1145/3503222.3507715.

[45] Albert T Schmitz, Nicolas PD Sawaya, Sonika Johri, and AY Matsuura. Graph optimization perspective for low-depth trotter-suzuki decomposition. *arXiv preprint arXiv:2103.08602*, 2021. doi: https://doi.org/10.48550/arXiv.2103.08602.

[46] Ewout Van Den Berg and Kristan Temme. Circuit optimization of hamiltonian simulation by simultaneous diagonalization of pauli clusters. *Quantum*, 4:322, 2020. doi: https://doi.org/10.22331/q-2020-09-12-322.

[47] Connor Powers, Lindsay Bassman, Thomas M. Linker, Ken ichi Nomura, Sahil Gulania, Rajiv K. Kalia, Aiichiro Nakano, and Priya Vashishta. Mistiqs: An open-source software for performing quantum dynamics simulations on quantum computers. *SoftwareX*, 14:100696, 2021. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2021.100696. URL https://www.sciencedirect.com/science/article/pii/S2352711021000418.

[48] Lindsay Bassman, Connor Powers, and Wibe A. De Jong. Arqtic: A full-stack software package for simulating materials on quantum computers. *ACM Transactions on Quantum Computing*, 3(3), jun 2022. doi: 10.1145/3511715. URL https://doi.org/10.1145/3511715.

[49] Sara Achour, Rahul Sarpeshkar, and Martin C. Rinard. Configuration synthesis for programmable analog devices with arco. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '16, page 177–193, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342612. doi: 10.1145/2908080.2908116. URL https://doi.org/10.1145/2908080.2908116.

[50] Sara Achour and Martin Rinard. Noise-aware dynamical system compilation for analog devices with legno. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 149–166, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371025. doi: 10.1145/3373376.3378449. URL https://doi.org/10.1145/3373376.3378449.

[51] Assa Auerbach. *Interacting electrons and quantum magnetism*. Springer Science & Business Media, 1998. doi: https://doi.org/10.1007/978-1-4612-0869-3.

[52] Bochen Tan and Jason Cong. Optimal layout synthesis for quantum computing. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020. doi: https://doi.org/10.1145/3400302.3415620.

[53] Sicun Gao, Soonho Kong, and Edmund M Clarke. dreal: An smt solver for nonlinear theories over the reals. In *Automated Deduction–CADE-24: 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings 24*, pages 208–214. Springer, 2013. doi: https://doi.org/10.1007/978-3-642-38574-2_14.

[54] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17 (3):261–272, 2020. doi: https://doi.org/10.1038/s41592-019-0686-2.

[55] Moein Malekakhlagh, Easwar Magesan, and David C McKay. First-principles analysis of cross-resonance gate operation. *Physical Review A*, 102(4):042605, 2020. doi: https://doi.org/10.1103/PhysRevA.102.042605.

[56] Thomas Alexander, Naoki Kanazawa, Daniel J Egger, Lauren Capelluto, Christopher J Wood, Ali Javadi-Abhari, and David C McKay. Qiskit pulse: Programming quantum computers through the cloud with pulses. *Quantum Science and Technology*, 5(4):044006, 2020. doi: 10.1088/2058-9565/aba404.

[57] Seth Lloyd. Universal quantum simulators. *Science*, 273(5278):1073–1078, 1996. doi: 10.1126/science.273.5278.1073.

[58] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[59] Andrew M. Childs, Dmitri Maslov, Yunseong Nam, Neil J. Ross, and Yuan Su. Toward the first quantum simulation with quantum speedup. *Proceedings of the National Academy of Sciences*, 115(38):9456–9461, 2018. doi: https://doi.org/10.1073/pnas.1801723115.

[60] Andrew M Childs, Yuan Su, Minh C Tran, Nathan Wiebe, and Shuchen Zhu. Theory of trotter error with commutator scaling. *Physical Review X*, 11(1):011020, 2021. doi: https://doi.org/10.1103/PhysRevX.11.011020.

[61] Prakash Murali, David C McKay, Margaret Martonosi, and Ali Javadi-Abhari. Software mitigation of crosstalk on noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1001–1016, 2020. doi: https://doi.org/10.1145/3373376.3378477.

[62] Felix Motzoi, Jay M Gambetta, Patrick Rebentrost, and Frank K Wilhelm. Simple pulses for elimination of leakage in weakly nonlinear qubits. *Physical review letters*, 103(11): 110501, 2009. doi: https://doi.org/10.1103/PhysRevLett.103.110501.

[63] David C McKay, Christopher J Wood, Sarah Sheldon, Jerry M Chow, and Jay M Gambetta. Efficient z gates for quantum computing. *Physical Review A*, 96(2):022330, 2017. doi: https://doi.org/10.1103/PhysRevA.96.022330.

[64] Nathan Earnest, Caroline Tornow, and Daniel J Egger. Pulse-efficient circuit transpilation for quantum applications on cross-resonance-based hardware. *Physical Review Research*, 3(4):043088, 2021. doi: https://doi.org/10.1103/PhysRevResearch.3.043088.

[65] Anders Sørensen and Klaus Mølmer. Entanglement and quantum computation with ions in thermal motion. *Physical Review A*, 62(2):022311, 2000. doi: https://doi.org/10.1103/PhysRevA.62.022311.

[66] Sean Greenaway, Adam Smith, Florian Mintert, and Daniel Malz. Analogue quantum simulation with fixed-frequency transmon qubits. *arXiv preprint arXiv:2211.16439*, 2022. doi: https://doi.org/10.48550/arXiv.2211.16439.

[67] John PT Stenger, Nicholas T Bronn, Daniel J Egger, and David Pekker. Simulating the dynamics of braiding of majorana zero modes using an ibm quantum computer. *Physical Review Research*, 3(3):033171, 2021. doi: 10.1103/PhysRevResearch.3.033171.

[68] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[69] Swamit S Tannu and Moinuddin K Qureshi. Mitigating measurement errors in quantum computers by exploiting state-dependent bias. In *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, pages 279–290, 2019. doi: https://doi.org/10.1145/3352460.3358265.

[70] Sam Kushnir, Jiaqi Leng, Yuxiang Peng, Lei Fan, and Xiaodi Wu. Qhdopt: A software for nonlinear optimization with quantum hamiltonian decent. 2024.

[71] Jiaqi Leng, Ethan Hickman, Joseph Li, and Xiaodi Wu. Quantum hamiltonian descent. *arXiv preprint arXiv:2303.01471*, 2023. doi: https://doi.org/10.48550/arXiv.2303.01471.

[72] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL `https://doi.org/10.7717/peerj-cs.103`.

[73] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–57, 2006.

[74] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL `https://www.gurobi.com`.

[75] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak 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, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[76] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, Peng Hu, Xiao-Yan Yang, Wei-Jun Zhang, Hao Li, Yuxuan Li, Xiao Jiang, Lin Gan, Guangwen Yang, Lixing You, Zhen Wang, Li Li, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020. ISSN 0036-8075. doi: 10.1126/science.abe8770. URL `https://science.sciencemag.org/content/370/6523/1460`.

[77] Y. Shi, P. Gokhale, P. Murali, J. M. Baker, C. Duckering, Y. Ding, N. C. Brown, C. Chamberland, A. Javadi-Abhari, A. W. Cross, D. I. Schuster, K. R. Brown, M. Martonosi, and F. T. Chong. Resource-efficient quantum computing by breaking abstractions. *Proceedings of the IEEE*, 108(8):1353–1370, 2020. doi: 10.1109/JPROC.2020.2994765.

[78] Alicia B. Magann, Christian Arenz, Matthew D. Grace, Tak-San Ho, Robert L. Kosut, Jarrod R. McClean, Herschel A. Rabitz, and Mohan Sarovar. From pulses to circuits and back again: A quantum optimal control perspective on variational quantum algorithms. *PRX Quantum*, 2:010101, Jan 2021. doi: 10.1103/PRXQuantum.2.010101. URL `https://link.aps.org/doi/10.1103/PRXQuantum.2.010101`.

[79] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5(1):1–7, 2014.

[80] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001, 2019.

[81] Aram W Harrow and Ashley Montanaro. Quantum computational supremacy. *Nature*, 549 (7671):203–209, 2017.

[82] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. Quantum circuit learning. *Physical Review A*, 98(3):032309, 2018.

[83] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh A. Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 2019.

[84] Shaopeng Zhu, Shih-Han Hung, Shouvanik Chakrabarti, and Xiaodi Wu. On the principles of differentiable quantum programming languages. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, page 272–285, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450376136. doi: 10.1145/3385412.3386011. URL `https://doi.org/10.1145/3385412.3386011`.

[85] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pages 3276–3285. PMLR, 2018.

[86] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *NeurIPS*, pages 6572–6583, 2018.

[87] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[88] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *ICLR*, 2020.

[89] William K Wootters and Wojciech H Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, 1982.

[90] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse problems*, 34(1):014004, 2017.

[91] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[92] Yulia Rubanova, Ricky T. Q. Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[93] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes. In *Advances in Neural Information Processing Systems*, volume 33, pages 3952–3963, 2020.

[94] Juntang Zhuang, Nicha C Dvornek, Sekhar Tatikonda, and James S Duncan. Mali: A memory efficient and reverse accurate integrator for neural odes. *International Conference on Learning Representations*, 2021.

[95] Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, 13, 2019.

[96] Filipe de Avila Belbute-Peres, Kevin A. Smith, Kelsey R. Allen, Josh Tenenbaum, and J. Zico Kolter. End-to-end differentiable physics for learning and control. In *Neural Information Processing Systems*, 2018.

[97] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Scalable differentiable physics for learning and control. In *ICML*, 2020.

[98] Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. Joint optimization of robot design and motion parameters using the implicit function theorem. In *Robotics: Science and Systems (RSS)*, 2017.

[99] Keenon Werling, Dalton Omens, Jeongseok Lee, Ionnis Exarchos, and C Karen Liu. Fast and feature-complete differentiable physics for articulated rigid bodies with contact. In *Robotics: Science and Systems (RSS)*, 2021.

[100] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Efficient differentiable simulation of articulated bodies. In *ICML*, 2021.

[101] Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Diffpd: Differentiable projective dynamics. *ACM Trans. Graph.*, 41 (2), 2021.

[102] Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. ADD: Analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics (TOG)*, 39(6), 2020.

[103] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Differentiable simulation of soft multi-body systems. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[104] Kiwon Um, Robert Brand, Yun (Raymond) Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers. In *Neural Information Processing Systems*, 2020.

[105] Nils Wandel, Michael Weinmann, and Reinhard Klein. Learning incompressible fluid dynamics from scratch – towards fast, differentiable fluid models that generalize. In *ICLR*, 2021.

[106] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control PDEs with differentiable physics. In *ICLR*, 2020.

[107] Tetsuya Takahashi, Junbang Liang, Yi-Ling Qiao, and Ming C Lin. Differentiable fluids with solid coupling for learning and control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6138–6146, 2021.

[108] Pingchuan Ma, Tao Du, John Z Zhang, Kui Wu, Andrew Spielberg, Robert K Katzschmann, and Wojciech Matusik. Diffaqua: A differentiable computational design pipeline for soft underwater swimmers with shape interpolation. *ACM Transactions on Graphics (TOG)*, 40(4):132, 2021.

[109] Eric Heiden, Miles Macklin, Yashraj S Narang, Dieter Fox, Animesh Garg, and Fabio Ramos. DiSECt: A Differentiable Simulation Engine for Autonomous Robotic Cutting. In *Proceedings of Robotics: Science and Systems*, 2021.

[110] Jatavallabhula Krishna Murthy, Miles Macklin, Florian Golemo, Vikram Voleti, Linda Petrini, Martin Weiss, Breandan Considine, Jérôme Parent-Lévesque, Kevin Xie, Kenny Erleben, et al. gradSim: Differentiable simulation for system identification and visuomotor control. In *ICLR*, 2021.

[111] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, 2017.

[112] Daoyi Dong and Ian R Petersen. Quantum control theory and applications: a survey. *IET Control Theory & Applications*, 4(12):2651–2671, 2010.

[113] Constantin Brif, Raj Chakrabarti, and Herschel Rabitz. Control of quantum phenomena: past, present and future. *New Journal of Physics*, 12(7):075008, 2010.

[114] Herschel Rabitz, Regina de Vivie-Riedle, Marcus Motzkus, and Karl Kompa. Whither the future of controlling quantum phenomena? *Science*, 288(5467):824–828, 2000.

[115] Mogens Dalgaard, Felix Motzoi, Jens Jakob Sørensen, and Jacob Sherson. Global optimization of quantum dynamics with alphazero deep exploration. *npj Quantum Information*, 6(1):1–9, 2020.

[116] Oinam Romesh Meitei, Bryan T. Gard, George S. Barron, David P. Pappas, Sophia E. Economou, Edwin Barnes, and Nicholas J. Mayhall. Gate-free state preparation for fast variational quantum eigensolver simulations. *npj Quantum Information*, 7(1):155, 2021.

[117] José P Palao and Ronnie Kosloff. Optimal control theory for unitary transformations. *Physical Review A*, 68(6):062308, 2003.

[118] Wusheng Zhu, Jair Botina, and Herschel Rabitz. Rapidly convergent iteration methods for quantum optimal control of population. *The Journal of Chemical Physics*, 108(5): 1953–1963, 1998.

[119] Wusheng Zhu and Herschel Rabitz. Noniterative algorithms for finding quantum optimal controls. *The Journal of chemical physics*, 110(15):7142–7152, 1999.

[120] Navin Khaneja, Timo Reiss, Cindie Kehlet, Thomas Schulte-Herbrüggen, and Steffen J Glaser. Optimal control of coupled spin dynamics: design of nmr pulse sequences by gradient ascent algorithms. *Journal of magnetic resonance*, 172(2):296–305, 2005.

[121] Tommaso Caneva, Tommaso Calarco, and Simone Montangero. Chopped random-basis quantum optimization. *Physical Review A*, 84(2):022326, 2011.

[122] Benjamin Dive, Alexander Pitchford, Florian Mintert, and Daniel Burgarth. In situ upgrade of quantum simulators to universal computers. *Quantum*, 2:80, 2018.

[123] Davide Castaldo, Marta Rosa, and Stefano Corni. Quantum optimal control with quantum computers: A hybrid algorithm featuring machine learning optimization. *Physical Review A*, 103(2):022613, 2021.

[124] Jun Li, Xiaodong Yang, Xinhua Peng, and Chang-Pu Sun. Hybrid quantum-classical approach to quantum optimal control. *Physical review letters*, 118(15):150503, 2017.

[125] Zhiding Liang, Hanrui Wang, Jinglei Cheng, Yongshan Ding, Hang Ren, Xuehai Qian, Song Han, Weiwen Jiang, and Yiyu Shi. Variational quantum pulse learning. *arXiv preprint arXiv:2203.17267*, 2022.

[126] Robert de Keijzer, Oliver Tse, and Servaas Kokkelmans. Pulse based variational quantum optimal control for hybrid quantum computing. *arXiv preprint arXiv:2202.08908*, 2022.

[127] Nikolaj Moll, Panagiotis Barkoutsos, Lev S Bishop, Jerry M Chow, Andrew Cross, Daniel J Egger, Stefan Filipp, Andreas Fuhrer, Jay M Gambetta, Marc Ganzhorn, et al. Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology*, 3(3):030503, 2018.

[128] Domenico d'Alessandro. *Introduction to quantum control and dynamics*. Chapman and Hall/CRC, 2007.

[129] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[130] Leonardo Banchi and Gavin E. Crooks. Measuring Analytic Gradients of General Quantum Evolution with the Stochastic Parameter Shift Rule. *Quantum*, 5:386, January 2021. ISSN 2521-327X. doi: 10.22331/q-2021-01-25-386. URL `https://doi.org/10.22331/q-2021-01-25-386`.

[131] Morten Kjaergaard, Mollie E. Schwartz, Jochen Braumüller, Philip Krantz, Joel I.-J. Wang, Simon Gustavsson, and William D. Oliver. Superconducting qubits: Current state of play. *Annual Review of Condensed Matter Physics*, 11(1):369–395, 2020. doi: 10.1146/annurev-conmatphys-031119-050605. URL `https://doi.org/10.1146/annurev-conmatphys-031119-050605`.

[132] He-Liang Huang, Dachao Wu, Daojin Fan, and Xiaobo Zhu. Superconducting quantum computing: a review. *Science China Information Sciences*, 63(8):1–32, 2020.

[133] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[134] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, USA, 2nd edition, 2012. ISBN 0521548233.

[135] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M Chow, and Jay M Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, 2017.

[136] Easwar Magesan and Jay M Gambetta. Effective hamiltonian models of the cross-resonance gate. *Physical Review A*, 101(5):052308, 2020.

[137] Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of IEEE international conference on evolutionary computation*, pages 312–317. IEEE, 1996.

[138] Dieter Kraft et al. A software package for sequential quadratic programming. 1988.

[139] Yuxiang Peng, Kesha Hietala, Runzhou Tao, Liyi Li, Robert Rand, Michael Hicks, and Xiaodi Wu. A formally certified end-to-end implementation of shor's factorization algorithm. *Proceedings of the National Academy of Sciences*, 120(21):e2218775120, 2023.

[140] Yuxiang Peng, Mingsheng Ying, and Xiaodi Wu. Algebraic reasoning of quantum programs via non-idempotent kleene algebra. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 657–670, 2022.

[141] Earl T. Campbell, Barbara M. Terhal, and Christophe Vuillot. Roads towards fault-tolerant universal quantum computation. *Nature*, 549(7671):172–179, 2017. doi: 10.1038/nature23460. URL `https://doi.org/10.1038/nature23460`.

[142] Barbara M. Terhal. Quantum error correction for quantum memories. *Rev. Mod. Phys.*, 87: 307–346, Apr 2015. doi: 10.1103/RevModPhys.87.307. URL `https://link.aps.org/doi/10.1103/RevModPhys.87.307`.

[143] Haskell B Curry. Functionality in combinatory logic. *Proceedings of the National Academy of Sciences of the United States of America*, 20(11):584, 1934.

[144] William Alvin Howard. The formulæ-as-types notion of construction. In Philippe De Groote, editor, *The Curry-Howard Isomorphism*. Academia, 1995.

[145] Tony Hoare. The verifying compiler: A grand challenge for computing research. *J. ACM*, 50(1):63–69, jan 2003. ISSN 0004-5411. doi: 10.1145/602382.602403. URL `https://doi.org/10.1145/602382.602403`.

[146] Richard A. De Millo, Richard J. Lipton, and Alan J. Perlis. Social processes and proofs of theorems and programs. *Commun. ACM*, 22(5):271–280, may 1979. ISSN 0001-0782. doi: 10.1145/359104.359106. URL `https://doi.org/10.1145/359104.359106`.

[147] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. SeL4: Formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, page 207–220, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605587523. doi: 10.1145/1629575.1629596. URL https://doi.org/10.1145/1629575.1629596.

[148] Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, July 2009. ISSN 0001-0782. doi: 10.1145/1538788.1538814. URL https://doi.org/10.1145/1538788.1538814.

[149] Xuejun Yang, Yang Chen, Eric Eide, and John Regehr. Finding and understanding bugs in c compilers. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, page 283–294, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306638. doi: 10.1145/1993498.1993532. URL https://doi.org/10.1145/1993498.1993532.

[150] Georges Gonthier et al. Formal proof–the four-color theorem. *Notices of the AMS*, 55(11): 1382–1393, 2008.

[151] Davide Castelvecchi. Mathematicians welcome computer-assisted proof in 'grand unification' theory. *Nature*, 595(7865):18–19, July 2021. doi: 10.1038/d41586-021-01627-.

[152] Kevin Hartnett. Proof assistant makes jump to big-league math. https://www.quantamagazine.org/lean-computer-program-confirms-peter-scholze-proof-20210728/, Jul 2021.

[153] The Coq Development Team. The coq proof assistant, September 2022. URL https://doi.org/10.5281/zenodo.7313584.

[154] Kesha Hietala, Robert Rand, Shih-Han Hung, Liyi Li, and Michael Hicks. Proving quantum programs correct. In *Proceedings of the Conference on Interative Theorem Proving (ITP)*, June 2021.

[155] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917855. doi: 10.1145/237814.237866. URL https://doi.org/10.1145/237814.237866.

[156] Jennifer Paykin, Robert Rand, and Steve Zdancewic. Qwire: A core language for quantum circuits. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, page 846–858, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346603. doi: 10.1145/3009837.3009894. URL https://doi.org/10.1145/3009837.3009894.

[157] Junyi Liu, Bohua Zhan, Shuling Wang, Shenggang Ying, Tao Liu, Yangjia Li, Mingsheng Ying, and Naijun Zhan. Formal verification of quantum algorithms using quantum hoare logic. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, pages 187–207, Cham, 2019. Springer International Publishing. ISBN 978-3-030-25543-5.

[158] Dominique Unruh. Quantum relational hoare logic. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019. doi: 10.1145/3290346. URL `https://doi.org/10.1145/3290346`.

[159] Liam Paninski. A coincidence-based test for uniformity given very sparsely sampled discrete data. *IEEE Transactions on Information Theory*, 54(10):4750–4755, 2008. doi: 10.1109/TIT.2008.928987.

[160] Donald E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., USA, 1997. ISBN 0201896834.

[161] Steven A Cuccaro, Thomas G Draper, Samuel A Kutin, and David Petrie Moulton. A new quantum ripple-carry addition circuit. *arXiv preprint quant-ph/0410184*, 2004.

[162] Antonio Lloris Ruiz, Encarnación Castillo Morales, Luis Parrilla Roure, and Antonio García Ríos. *Algebraic circuits*. Springer, 2014.

[163] Thomas G Draper. Addition on a quantum computer. *arXiv preprint quant-ph/0008033*, 2000.

[164] Thomas G Draper, Samuel A Kutin, Eric M Rains, and Krysta M Svore. A logarithmic-depth quantum carry-lookahead adder. *arXiv preprint quant-ph/0406142*, 2004.

[165] Rodney Van Meter and Kohei M Itoh. Fast quantum modular exponentiation. *Physical Review A*, 71(5):052320, 2005.

[166] Archimedes Pavlidis and Dimitris Gizopoulos. Fast quantum modular exponentiation architecture for shor's factorization algorithm. *arXiv preprint arXiv:1207.0511*, 2012.

[167] Craig Gidney and Martin Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, April 2021. ISSN 2521-327X. doi: 10.22331/q-2021-04-15-433. URL `https://doi.org/10.22331/q-2021-04-15-433`.

[168] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford, fourth edition, 1975.

[169] Andrej Bauer. Five stages of accepting constructive mathematics. *Bulletin of the American Mathematical Society*, 2016. doi: https://doi.org/10.1090/bull/1556.

[170] Daniel De Rauglaudre. Coq proof of the euler product formula for the riemann zeta function. `https://github.com/roglo/coq_euler_prod_form`, 2020.

[171] J. Barkley Rosser and Lowell Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois J. Math.*, 6(1):64–94, 03 1962. doi: 10.1215/ijm/1255631807. URL `https://doi.org/10.1215/ijm/1255631807`.

[172] Andrew W Cross, Lev S Bishop, John A Smolin, and Jay M Gambetta. Open quantum assembly language. *arXiv preprint arXiv:1707.03429*, 2017.

[173] Program extraction. `https://coq.inria.fr/refman/addendum/extraction.html`, 2021. Accessed: 2021-09-24.

[174] Library Coq.Reals.Reals. `https://coq.inria.fr/library/Coq.Reals.Reals.html`. Accessed: 2021-09-24.

[175] Sylvie Boldo and Guillaume Melquiond. Flocq: A unified library for proving floating-point algorithms in coq. In *2011 IEEE 20th Symposium on Computer Arithmetic*, pages 243–252, 2011. doi: 10.1109/ARITH.2011.40.

[176] JKQ DDSIM – a quantum circuit simulator based on decision diagrams written in C++. `https://github.com/iic-jku/ddsim`, 2021.

[177] S. C. Kleene. *Representation of Events in Nerve Nets and Finite Automata*, pages 3 – 42. Princeton University Press, Princeton, 1956. doi: https://doi.org/10.1515/9781400882618-002. URL `https://princetonup.degruyter.com/view/book/9781400882618/10.1515/9781400882618-002.xml`.

[178] Dexter Kozen and Frederick Smith. Kleene algebra with tests: Completeness and decidability. In D. van Dalen and M. Bezem, editors, *Proc. 10th Int. Workshop Computer Science Logic (CSL'96)*, volume 1258 of *Lecture Notes in Computer Science*, pages 244–259, Utrecht, The Netherlands, September 1996. Springer-Verlag.

[179] Dexter Kozen and Maria-Cristina Patron. Certification of compiler optimizations using Kleene algebra with tests. In John Lloyd, Veronica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luis Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Proc. 1st Int. Conf. Computational Logic (CL2000)*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 568–582, London, July 2000. Springer-Verlag.

[180] Dexter Kozen. On Hoare logic and Kleene algebra with tests. *Trans. Computational Logic*, 1(1):60–76, July 2000.

[181] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: Semantic foundations for networks. In *Proc. 41st ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages (POPL'14)*, pages 113–126, San Diego, California, USA, January 2014. ACM.

[182] Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. A coalgebraic decision procedure for NetKAT. In *Proc. 42nd ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages (POPL'15)*, pages 343–355, Mumbai, India, January 2015. ACM.

[183] Steffen Smolka, Nate Foster, Justin Hsu, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded kleene algebra with tests: Verification of uninterpreted programs in nearly linear time. *Proc. ACM Program. Lang.*, 4(POPL), December 2019. doi: 10.1145/3371129. URL `https://doi.org/10.1145/3371129`.

[184] Ernie Cohen, Dexter Kozen, and Frederick Smith. The complexity of Kleene algebra with tests. Technical Report TR96-1598, Computer Science Department, Cornell University, July 1996.

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

[186] Jennifer Paykin, Robert Rand, and Steve Zdancewic. QWIRE: A core language for quantum circuits. POPL 2017, page 846–858, 2017. ISBN 9781450346603.

[187] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q#: Enabling scalable quantum computing and development with a high-level DSL. In *RWDSL*, 2018.

[188] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, F Jose Cabrera-Hernández, Jorge Carballo-Franquis, Adrian Chen, Chun-Fu Chen, et al. Qiskit: An open-source framework for quantum computing. *Accessed on: Mar*, 16, 2019. doi: 10.5281/zenodo.2562110.

[189] Google. `https://github.com/quantumlib/Cirq`, 2018.

[190] Rigetti. `https://www.rigetti.com/forest`, 2018.

[191] Bernhard Ömer. *Structured Quantum Programming*. PhD thesis, Vienna University of Technology, 2003.

[192] Jeff W. Sanders and Paolo Zuliani. Quantum programming. In *MPC*, 2000.

[193] Amr Sabry. Modeling quantum computing in haskell. In *The Haskell Workshop*, 2003.

[194] Peter Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4), 2004.

[195] Jonathan Grattage. A functional quantum programming language. In *LICS*, 2005.

[196] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10): 576–580, October 1969. ISSN 0001-0782.

[197] Olivier Brunet and Philippe Jorrand. Dynamic quantum logic for quantum programs. *International Journal of Quantum Information*, 2(1), 2004.

[198] Rohit Chadha, Paulo Mateus, and Amílcar Sernadas. Reasoning about imperative quantum programs. *Electronic Notes in Theoretical Computer Science*, 158, 2006.

[199] Alexandru Baltag and Sonja Smets. Quantum logic as a dynamic logic. *Synthese*, 179(2): 285–306, 2011.

[200] Yuan Feng, Runyao Duan, Zhengfeng Ji, and Mingsheng Ying. Proof rules for the correctness of quantum programs. *Theoretical Computer Science*, 386(1-2), 2007.

[201] Yoshihiko Kakutani. A logic for formal verification of quantum programs. In *ASIAN 2009*, pages 79–93, 2009.

[202] Mingsheng Ying. Floyd–Hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems*, 33(6), 2011.

[203] Ellie D'Hondt and Prakash Panangaden. Quantum weakest preconditions. *Mathematical Structures in Computer Science*, 16(3), 2006.

[204] Mingsheng Ying, Shenggang Ying, and Xiaodi Wu. Invariants of quantum programs: Characterisations and generation. POPL 2017, page 818–832, 2017.

[205] Yangjia Li and Mingsheng Ying. Algorithmic analysis of termination problems for quantum programs. 2(POPL), December 2017.

[206] Li Zhou, Nengkun Yu, and Mingsheng Ying. An applied quantum Hoare logic. PLDI 2019, page 1149–1162, 2019.

[207] Peter Selinger. A brief survey of quantum programming languages. In *FLOPS*, 2004.

[208] Simon J. Gay. Quantum programming languages: Survey and bibliography. *Mathematical Structures in Computer Science*, 16(4), 2006.

[209] Mingsheng Ying. Toward automatic verification of quantum programs. *Formal Aspects of Computing*, 31(1):3–25, Feb 2019.

[210] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194 – 211, 1979. ISSN 0022-0000. doi: https://doi.org/10.1016/0022-0000(79)90046-1. URL `http://www.sciencedirect.com/science/article/pii/0022000079900461`.

[211] Mike Mislove. On combining probability and nondeterminism. *Electronic Notes in Theoretical Computer Science*, 162:261 – 265, 2006. Proceedings of the Workshop Essays on Algebraic Process Calculi (APC 25).

[212] Daniele Varacca and Glynn Winskel. Distributing probability over non-determinism. *Mathematical Structures in Computer Science*, 16(1):87–113, 2006. doi: 10.1017/S0960129505005074.

[213] Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. Probabilistic NetKAT. In Peter Thiemann, editor, *25th European Symposium on Programming (ESOP 2016)*, volume 9632 of *Lecture Notes in Computer Science*, pages 282–309, Eindhoven, The Netherlands, April 2016. Springer. doi: 10.1007/978-3-662-49498-1_12.

[214] Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*. Springer-Verlag, Berlin, Heidelberg, 1985. ISBN 3540137165.

[215] Zoltán Ésik and Werner Kuich. Inductive *-semirings. *Theoretical Computer Science*, 324 (1):3–33, 2004.

[216] David J Foulis and Mary K Bennett. Effect algebras and unsharp quantum logics. *Foundations of physics*, 24(10):1331–1352, 1994.

[217] Dominique Unruh. Quantum relational Hoare logic. 3(POPL), January 2019.

[218] Gilles Barthe, Justin Hsu, Mingsheng Ying, Nengkun Yu, and Li Zhou. Relational proofs for quantum programs. *Proc. ACM Program. Lang.*, 4(POPL), December 2019. doi: 10.1145/3371089. URL `https://doi.org/10.1145/3371089`.

[219] Nengkun Yu and Jens Palsberg. Quantum abstract interpretation. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, page 542–558, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383912. doi: 10.1145/3453483.3454061. URL `https://doi.org/10.1145/3453483.3454061`.

[220] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. A verified optimizer for quantum circuits. *arXiv preprint arXiv:1912.02250*, 2019.

[221] Sam Staton. Algebraic effects, linearity, and quantum programming languages. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, page 395–406, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450333009. doi: 10.1145/2676726.2676999. URL `https://doi.org/10.1145/2676726.2676999`.

[222] M.P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2):245 – 270, 1961. ISSN 0019-9958. doi: https://doi.org/10.1016/S0019-9958(61)80020-X. URL `http://www.sciencedirect.com/science/article/pii/S001999586180020X`.

[223] Jean Berstel and Christophe Reutenauer. *Noncommutative rational series with applications*, volume 137. Cambridge University Press, 2011.

[224] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of weighted automata*. Springer Science & Business Media, 2009.

[225] Marie-Pierre Béal, Sylvain Lombardy, and Jacques Sakarovitch. On the equivalence of $\mathbb{Z}$-automata. In *International Colloquium on Automata, Languages, and Programming*, pages 397–409. Springer, 2005.

[226] Marie-Pierre Béal, Sylvain Lombardy, and Jacques Sakarovitch. Conjugacy and equivalence of weighted automata and functional transducers. In *International Computer Science Symposium in Russia*, pages 58–69. Springer, 2006.

[227] Stephen L Bloom and Zoltán Ésik. Axiomatizing rational power series over natural numbers. *Information and Computation*, 207(7):793–811, 2009.

[228] Larry J Stockmeyer and Albert R Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 1–9, 1973.

[229] Samuel Eilenberg. *Automata, languages, and machines*. Academic press, 1974.

[230] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. doi: 10.1017/CBO9780511976667.

[231] John Watrous. *The Theory of Quantum Information*. Cambridge University Press, 2018. doi: 10.1017/9781316848142.

[232] Karl Kraus, Arno Böhm, John D Dollard, and WH Wootters. States, effects, and operations: fundamental notions of quantum theory. lectures in mathematical physics at the university of texas at austin. *Lecture notes in physics*, 190, 1983.

[233] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886): 802–803, 1982.

[234] Mingsheng Ying. *Foundations of Quantum Programming*. Morgan Kaufmann, 2016.

[235] Andrew M. Childs, Dmitri Maslov, Yunseong Nam, Neil J. Ross, and Yuan Su. Toward the first quantum simulation with quantum speedup. *Proceedings of the National Academy of Sciences*, 115(38):9456–9461, 2018. ISSN 0027-8424.

[236] Guang Hao Low and Isaac L Chuang. Optimal hamiltonian simulation by quantum signal processing. *Physical review letters*, 118(1):010501, 2017. doi: 10.1103/PhysRevLett.118.010501.

[237] Corrado Böhm and Giuseppe Jacopini. Flow diagrams, turing machines and languages with only two formation rules. *Commun. ACM*, 9(5):366–371, May 1966. ISSN 0001-0782. doi: 10.1145/355592.365646. URL `https://doi.org/10.1145/355592.365646`.

[238] Nengkun Yu. Quantum Temporal Logic. *arXiv e-prints*, art. arXiv:1908.00158, July 2019.