University of Verona

Department of Computer Science

graduate school of natural sciences and engineering

doctoral program in computer science

cycle 38

Formal Semantics and Analysis of Quantum Programs

S.S.D. INF/01

Coordinator: Umberto Castellani

Advisors: Alessandra Di Pierro & Isabella Mastroeni

Doctoral Student: Nicola Assolini

Formal Semantics and Analysis of Quantum Programs - Nicola Assolini
Tesi di Dottorato
Verona, December 06, 2025

Sommario

In seguito alle esplorazioni teoriche della fine degli anni Novanta, le tecnologie quantistiche hanno conosciuto un rapido sviluppo, sostenuto da ingenti investimenti pubblici e privati (ad esempio da parte di governi, dell'Unione Europea, di D-Wave, IBM e Google). L'avvento dei dispositivi quantistici programmabili ha rafforzato la percezione che stiamo entrando in una *era quantistica*. La programmazione di tali sistemi sta diventando una competenza fondamentale per gli sviluppatori del futuro, e l'ingegneria del software quantistico si configura come una nuova frontiera per la teoria dei linguaggi di programmazione e i metodi formali. Nonostante questi progressi, l'attuale ecosistema rimane ancora immaturo: sebbene esistano diversi linguaggi di alto livello (ad esempio Q# [Svo+18], Silq [Bic+20], QWIRE [PRZ17]) e framework (ad esempio Qiskit [Ale+19], Cirq [Cir25], t|ket⟩ [Siv+20]), il ragionamento sulle *proprietà dei programmi* è tuttora poco compreso. A differenza dei programmi classici, le computazioni quantistiche evolvono in spazi vettoriali complessi e obbediscono ai principi della meccanica quantistica; fenomeni come la sovrapposizione, l'entanglement, il teorema di non-clonazione [NC10, Ch. 12] e la misura implicita [NC10, Sec. 4.4] introducono comportamenti intrinsecamente non classici che le tecniche di analisi e verifica devono essere in grado di catturare.

L'interpretazione astratta [CC77; Cou21b] e l'analisi statica [RY20; NNH99] offrono potenti strumenti per il ragionamento formale sui programmi classici, ma richiedono profonde adattamenti al contesto quantistico. La computazione quantistica modifica il modo in cui si modellano gli stati e il flusso di controllo di un programma: le analisi classiche non possono essere semplicemente riutilizzate, ma devono essere estese per catturare gli effetti specifici del calcolo quantistico e, quando necessario, arricchite con nuovi domini astratti per rappresentare proprietà tipicamente quantistiche, come l'entanglement. Inoltre, la sovrapposizione all'interno delle strutture di controllo introduce sfide semantiche inedite, poiché il flusso di controllo stesso può evolvere in modo coerente—un aspetto privo di analoghi nel mondo classico.

Dopo aver passato in rassegna i lavori precedenti sull'applicazione dell'interpretazione astratta e dell'analisi statica ai programmi quantistici (Chapter 3), questa tesi introduce tre nuovi approcci basati sull'interpretazione astratta che affrontano problemi distinti—*uncomputation*, *entanglement* e *robustezza dei circuiti quantistici variazionali*—insieme a un quadro semantico per il ragionamento sul controllo quantistico.

Il primo contributo (Chapter 4) definisce un quadro solido per il ragionamento sull'uso e sul ciclo di vita delle variabili quantistiche. A causa del teorema di non-clonazione e dell'entanglement, operazioni come la copia o la cancellazione devono essere reinterpretate: le variabili non utilizzate devono essere esplicitamente reimpostate per evitare misurazioni implicite. Due analisi di flusso dei dati complementari—una *analisi di consumo* diretta e una *analisi di uncomputation* inversa—rilassano i vincoli imposti dai tipi lineari garantendo al contempo la correttezza, e consentono l'inserimento automatico delle operazioni `discard`.

Su queste basi, il secondo contributo (Chapter 5) sviluppa un quadro di interpretazione astratta per il ragionamento sull'entanglement. Un dominio raffinato cattura le relazioni di inseparabilità e le caratteristiche qualitative degli stati dei qubit (ad esempio, classicità, sovrapposizione, fase), consentendo di tracciare in modo corretto l'entanglement senza esplosione esponenziale e supportando ottimizzazioni e analisi consapevoli dell'entanglement.

Il terzo contributo (Chapter 6) estende la verifica formale ai circuiti quantistici variazionali (VQC), centrali per le applicazioni a breve termine. La tesi formalizza il *problema della verifica della robustezza per i VQC (RVVQC)*, ne dimostra la NP-durezza, e definisce sia una semantica concreta sia una semantica astratta per un linguaggio dedicato ai VQC. Strategie di raffinamento come il *clipping*, l'esecuzione simbolica e la partizione del dominio migliorano la precisione delle astrazioni intervallari, producendo una pipeline di verifica pratica in grado di certificare garanzie di robustezza per i VQC.

Infine, l'ultimo contributo (Chapter 7) propone una semantica denotazionale per i cicli quantistici. Poiché il controllo basato su misura non è ammesso e la semantica unitaria non riesce a catturare comportamenti infiniti, viene introdotta una *semantica lineare* basata su operatori lineari limitati. Essa garantisce la convergenza e approssima in modo corretto l'evoluzione concreta, fornendo una base solida per il ragionamento sulla terminazione e la divergenza nei programmi quantistici.

Abstract

Following the theoretical explorations of the late 1990s, quantum technologies have rapidly advanced, driven by major public and private investments (e.g., from governments, the EU, D-Wave, IBM, Google). The advent of programmable quantum devices has reinforced the perception that we are entering a *quantum era*. Programming such systems is becoming a key skill, and quantum software engineering is emerging as a new frontier for programming language theory and formal methods. Despite this progress, the ecosystem remains immature: several high-level languages (e.g., Q# [Svo+18], Silq [Bic+20], QWIRE [PRZ17]) and frameworks (e.g., Qiskit [Ale+19], Cirq [Cir25], t|ket⟩ [Siv+20]) exist, yet reasoning about *quantum program properties* is still poorly understood. Unlike classical programs, quantum computations evolve over vector spaces and obey quantum-mechanical principles; phenomena such as superposition, entanglement, no-cloning [NC10, Ch. 12], and implicit measurement [NC10, Ch. 4] introduce behaviors that analyses must capture.

Abstract interpretation [CC77; Cou21b] and static analysis [RY20; NNH99] provide powerful reasoning tools but require substantial adaptation if applied to quantum programming languages. Similarly, as quantum computation reshapes program states and control flow, classical analyses techniques cannot be applied directly; new domains are needed for quantum-specific properties such as entanglement, and coherent control introduces semantic challenges with no classical analogue.

After reviewing prior work (Chapter 3), this thesis presents three new abstract-interpretation-based approaches addressing *uncomputation*, *entanglement*, and *robustness of variational quantum circuits*, together with a semantic framework for quantum control flow.

The first contribution (Chapter 4) defines a framework for reasoning about the use and lifetime of quantum variables. Because of no-cloning and entanglement, copying and deletion require a careful treatment; in particular, unused variables must be reset before being discarded, so as to avoid implicit measurement. Two data-flow analyses—a forward *consuming analysis*

and a backward *uncomputation analysis*—relax linear typing while ensuring soundness and enabling automatic insertion of `discard` operations.

The second contribution (Chapter 5) develops an abstract domain for reasoning about entanglement. It captures inseparability relations and qualitative features of qubit states (superposition, phase), enabling sound tracking without exponential blow-up and supporting entanglement-aware optimization.

The third contribution (Chapter 6) extends formal verification to variational quantum circuits (VQCs). The thesis formalizes the *robustness verification problem for VQCs*, proves its NP-hardness, and defines concrete and abstract semantics for a VQC language. Refinement strategies—clipping, symbolic execution, and domain partitioning—improve interval precision, yielding a practical pipeline that certifies robustness guarantees.

Finally, Chapter 7 introduces a denotational semantics for quantum loops. Since measurement-based control is disallowed and unitary semantics cannot model unbounded behaviors, we introduce a *linear semantics* based on bounded operators to ensure convergence at the cost of a sound approximation of the concrete evolution, thus providing a foundation for reasoning about termination and divergence.

List of publications and Preprints

This document builds upon the following works:

- *Static Analysis of Quantum Programs*. **Nicola Assolini**, Alessandra Di Pierro, Isabella Mastroeni. In: Static Analysis - 31st International Symposium, SAS, 2024 [ADM24b].
- *Abstracting Entanglement*. **Nicola Assolini**, Alessandra Di Pierro, Isabella Mastroeni. In: Proceedings of the 10th ACM SIGPLAN International Workshop on Numerical and Symbolic Abstract Domains, NSAD, 2024 [ADM24a].
- *A Static Analysis for High-Level Quantum Programming Languages*. **Nicola Assolini**, Alessandra Di Pierro, Isabella Mastroeni. In: Verification, Model Checking, and Abstract Interpretation - 26th International Conference, VMCAI, 2025 [ADM25].
- *A Semantics for Quantum Loops*. **Nicola Assolini**, Alessandra Di Pierro. arXiv, 2025 [AP25].
- *Formal Verification of Variational Quantum Circuits*. **Nicola Assolini**, Luca Marzari, Isabella Mastroeni, Alessandra Di Pierro. arXiv, 2025 [AMM25]
- *Challenges in Quantum Programs Analysis*. **Nicola Assolini**, Alessandra Di Pierro, Isabella Mastroeni. In: International Journal on Software Tools for Technology Transfer [APM26]

Table of Contents

List of Figures

Chapter 1

Introduction

Following the initial theoretical explorations in the late 1990s, the field of quantum technologies has undergone an extraordinary acceleration, recently bolstered by substantial investments from both the public sector (e.g., national governments, the European Union) and the private sector (notably D-Wave, IBM, and Google). This progress, together with the realization of programmable quantum devices, justifies the widespread perception that we are entering a *quantum era*. In this context, learning how to program quantum computers is emerging as a critical skill for the next generation of software developers, and quantum software engineering is increasingly recognized as a new frontier for programming language theory and formal methods.

Despite these remarkable advancements, the current quantum programming ecosystem remains at an early stage of maturity. While several high-level languages (such as Q# [Svo+18], Silq [Bic+20], and QWIRE [PRZ17]) and frameworks (such as Qiskit [Ale+19], Cirq [Cir25], and t|ket⟩ [Siv+20]) have been proposed, several fundamental aspects of *reasoning about program properties* remain to be explored. The challenges of analyzing and verifying quantum programs stem from the fact that, unlike classical programs, quantum computations evolve over complex vector spaces and are governed by the principles of quantum mechanics. Phenomena such as superposition, entanglement, the no-cloning theorem [NC10, Chapter 12], and the principle of implicit measurement [NC10, Chapter 4] introduce inherently non-classical behaviors that must be taken into account when designing analysis and verification techniques.

Abstract interpretation [CC77; Cou21b] and static analysis [RY20; NNH99] are powerful

frameworks for formal reasoning about classical programs. Their direct application to quantum programs would, however, not be able to capture quantum-specific phenomena, which require the definition of new abstract domains to reason about quantum properties such as entanglement. Moreover, the presence of superposition within quantum control structures introduces novel semantic challenges, since control flow itself evolves in a quantum-coherent way—an aspect that has no classical counterpart.

After reviewing the literature on the applications of abstract interpretation and static analysis to quantum settings, this thesis presents three novel abstract-interpretation-based approaches addressing distinct quantum problems—*uncomputation*, *entanglement*, and *robustness of variational quantum circuits*—as well as a new semantic framework for reasoning about quantum programs characterized by quantum control flow.

In the following, we summarize the content of each chapter of this thesis.

Challenges in Quantum Program Analysis (Chapter 3). This chapter surveys the literature on static analyses for quantum programs, with the aim of systematizing existing approaches and identifying common patterns, differences, and open challenges. As in the classical setting, the choice of analysis technique depends on the property of interest. A first class of approaches, called *Flow-Based Analyses*, focuses on properties of quantum variables and adapts classical dataflow techniques to the quantum setting. These analyses track variable usage and control flow to ensure correctness conditions specific to quantum computation—such as preventing illegal duplication or discarding of quantum data, in accordance with the no-cloning theorem and implicit measurement principles. A second class, referred to as *Domain-Based Analyses*, targets quantum state properties such as entanglement. Here, abstract interpretation provides a mathematical framework to approximate the exponentially large Hilbert spaces of quantum programs, enabling sound and efficient reasoning about their behavior.

Static analysis of quantum variables and uncomputation (Chapter 4). The chapter *Static Analysis of Quantum Programs* introduces a sound framework for reasoning about the use and lifetime of quantum variables. Because of the no-cloning theorem and the presence of entanglement, traditional notions of assignment, copying, or deletion do not directly apply. In particular, unused quantum variables must be explicitly reset (*uncomputed*) to avoid implicit measurement effects that could propagate through the system. The proposed approach defines two complementary dataflow analyses: a forward *consuming analysis*, ensuring that variables are used at most once, and a backward *uncomputation analysis*, identifying variables that can be safely discarded. This framework relaxes the strict constraints imposed by other approaches based on linear type systems while retaining soundness, allowing the compiler to

automatically insert the necessary `discard` operations. In doing so, it strikes a balance between the expressive power of high-level quantum programming and the operational constraints imposed by quantum mechanics.

Static analysis of entanglement (Chapter 5). Building on the previous framework, the chapter *A Static Analysis of Entanglement* develops an abstract interpretation approach to reason about one of the most distinctive quantum phenomena: entanglement. The analysis introduces a refined abstract domain that simultaneously captures inseparability relations among qubits and qualitative information about their abstract states—such as whether they are classical, in uniform superposition, or characterized by specific phase relations. By representing entanglement in a domain-theoretic setting, this thesis provides a formal and computationally feasible way to track the evolution of entanglement during program execution. The resulting analysis is sound and improves the precision of previous approaches, avoiding the exponential blow-up associated with explicit state representations. This contribution bridges the gap between quantum semantics and compiler optimization, enabling entanglement-aware reasoning and transformations.

Formal verification of variational quantum circuits (Chapter 6). The chapter *Formal Verification of Variational Quantum Circuits (VQCs)* extends formal verification techniques to hybrid quantum-classical algorithms, which represent the cornerstone of near-term quantum applications. Variational circuits, widely used in quantum machine learning and optimization, are known to be vulnerable to small input perturbations, similarly to classical neural networks. In this thesis we formally define and study the *robustness verification problem for VQCs (RVVQC)* within a semantic framework grounded in abstract interpretation. After giving a formal proof of the NP-hardness of the verification problem, we introduce both concrete and abstract semantics for an interval-based analysis of a dedicated VQC language. To mitigate the incompleteness of non-relational interval abstractions, we propose several refinement strategies, such as clipping, symbolic execution and iterative domain partitioning, which allow for a more precise characterization of robustness. The resulting verification pipeline represents the first interval-based tool capable of certifying provable bounds on admissible input perturbations for VQCs, advancing the field toward trustworthy hybrid algorithms.

Denotational semantics for quantum loops (Chapter 7). The final chapter *A Denotational Semantics for Quantum Loops* addresses one of the most fundamental challenges in defining the semantics of quantum programming languages: providing a mathematically sound treatment of iterative constructs. In the quantum setting, control flow cannot rely on measurement, since observing a quantum guard irreversibly alters the program state.

Moreover, the standard *unitary semantics*, which precisely models the physical evolution of the system, fails to capture infinite computations, as the sequence of unitary operators generated by successive loop iterations does not converge. To overcome this limitation, we propose an approach which replaces unitaries with bounded linear operators, introducing a *linear semantics* that guarantees convergence at the cost of an under-approximation of the concrete (unitary) behavior. This semantics captures both finite and infinite computations, distinguishing, in the limit, between terminating and non-terminating executions, and offering a well-defined mathematical foundation for reasoning about quantum loops.

To summarize, this thesis contributes to advancing the theoretical and methodological foundations of quantum program analysis. By adapting and extending classical principles such as denotational semantics and abstract interpretation to the quantum realm, we provide the basis for rigorous reasoning about quantum computations, paving the way for the next generation of quantum software tools—ones that are not only expressive and efficient, but also formally verifiable and robust against the intrinsic complexity of quantum behavior.

Chapter 2

Preliminaries

In this chapter, we provide the theoretical background required for the rest of the thesis. In Section 2.1, we recall the foundations of *abstract interpretation*, following the seminal work of Cousot and Cousot [CC77]. We introduce the notions of concrete and abstract domains, together with the adjoint relation between them, formalized through Galois connections. In Section 2.2, we review the mathematical foundations of *quantum computation*, summarizing the key notions and principles that will play a central role in the developments of this work. For a more detailed exposition, we refer the reader to [NC10; Lin22; HZ08].

2.1 Abstract Interpretation

In this section, we introduce the basic algebraic notions that will be used throughout the thesis. We recall the mathematical background on sets and relations [JW96] and then we present the framework of abstract interpretation [CC77; CC79], summarizing the main characterizations of abstract domains that appear in the literature.

2.1.1 Basic Notion

We first recall some basic definitions about sets and relations, as well as structured forms of ordered sets.

Definition 2.1 (Equivalence relation)**.** *Let R be a binary relation on a set A. We say that R is an* equivalence relation *if the following conditions hold:*

(1) $\forall x \in A.\ \langle x, x\rangle \in R$ *(reflexivity);*

(2) $\forall x, y \in A.\ \langle x, y\rangle \in R \ \Rightarrow\ \langle y, x\rangle \in R$ *(symmetry);*
(3) $\forall x, y, z \in A.\ \langle x, y\rangle \in R \wedge \langle y, z\rangle \in R \ \Rightarrow\ \langle x, z\rangle \in R$ *(transitivity).*

Whenever $\langle x, y\rangle \in R$*, we also write* xRy*. If* R *fails the reflexive condition, then it is said to be a* partial equivalence relation (PER)*.*

If A *is equipped with an equivalence relation* R*, we consider, for each* $x \in A$*, the subset*

$$A_x = \{\, y \in A \mid yRx \,\}.$$

These sets are called equivalence classes *of* A *with respect to* R*, and they are usually denoted by* $[x]_R$*.*

Definition 2.2 (Partial order)**.** *Let* P *be a set. A* partial order *on* P *is a binary relation* $\leq$ *such that, for all* $x, y, z \in P$*:*

(1) $x \leq x$ *(reflexivity);*
(2) $x \leq y \wedge y \leq x \ \Rightarrow\ x = y$ *(antisymmetry);*
(3) $x \leq y \wedge y \leq z \ \Rightarrow\ x \leq z$ *(transitivity).*

An order relation R *on* P *is said to be* strict *if it is irreflexive, i.e.,* $\forall x \in P.\ \langle x, x\rangle \notin R$*. A partial order on* P *is said to be a* linear order *if every two elements of* P *are comparable.*

Definition 2.3 (Poset)**.** *A* partially ordered set (poset) *is a pair* $\langle P, \leq_P\rangle$ *where* P *is a set and* $\leq_P$ *is a partial order on* P*. If all pairs of elements in* P *are comparable, then* P *is said to be* totally ordered*, or a* chain*.*

Definition 2.4 (Directed set)**.** *Let* P *be a poset. We say that* P *is a* directed set *if each non-empty finite subset of* P *has a least upper bound in* P*. For example, every chain is a directed set.*

Definition 2.5 (cpo)**.** *A* complete partial order (cpo) *is a poset* $\langle P, \leq_P\rangle$ *such that:*

- P *contains a least element* $\bot$*;*
- *every directed set* $D \subseteq P$ *has a least upper bound* $\bigvee D \in P$*.*

Example 2.1. Every finite poset is a cpo. The set of natural numbers $\mathbb{N}$ with the usual order is a cpo only if we add the limit element ω, since otherwise the set lacks the least upper bound of all natural numbers.

Proposition 2.1. *A poset* $\langle P, \leq_P\rangle$ *is a cpo if and only if each chain in* P *has a least upper bound.*

2.1.2 Abstract Domains

We briefly recall the definition of abstract domains in the sense of Cousot and Cousot [CC77]. Let $(C, \leq_C)$ be a complete lattice (or a cpo), representing the *concrete domain*, and let $(A, \leq_A)$ be a poset of abstract elements, ordered by their precision: $x \leq_A y$ means that x is more precise than y in describing a computational property. The core of abstract interpretation lies in the adjoint relation between the abstract and concrete domains, known as a *Galois connection*.

Definition 2.6 (Galois connection)**.** *Let $(A, \leq_A)$ and $(C, \leq_C)$ be two posets, and let $\alpha : C \to A$ and $\gamma : A \to C$ be two monotone functions. We say that $(C, \leq_C) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \leq_A)$ is a* Galois connection*(GC) if, for all $a \in A$ and $c \in C$,*

$$\alpha(c) \leq_A a \iff c \leq_C \gamma(a).$$

Here, α is called the abstraction function *and γ the* concretization function*.*

Galois connections satisfy the following properties:

- $\gamma \circ \alpha$ is *extensive*: $\forall c \in C,\ c \leq_C \gamma(\alpha(c))$.
- $\alpha \circ \gamma$ is *reductive*: $\forall a \in A,\ \alpha(\gamma(a)) \leq_A a$.

A stronger notion is that of *Galois insertion*(GI).

Definition 2.7 (Galois insertion)**.** *A Galois connection $(C, \leq_C) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \leq_A)$ is a* Galois insertion *if $\alpha \circ \gamma$ is the indentity in A.*

In this case, α is surjective and γ is injective, and the order structure of C is faithfully transferred to A. Moreover, if C is a complete lattice, then A is also a complete lattice.

Finally, Galois connections compose as follows.

Proposition 2.2. *If $(C, \leq_C) \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} (B, \leq_B)$ and $(B, \leq_B) \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} (A, \leq_A)$, then $(C, \leq_C) \underset{\alpha_2 \circ \alpha_1}{\overset{\gamma_1 \circ \gamma_2}{\leftrightarrows}} (A, \leq_A)$ is also a Galois connection.*

2.1.3 Soundness and Completeness

In this section, we introduce the notions of *soundness* and *completeness* of an abstract domain with respect to a given concrete function or operation. The very essence of abstract interpretation lies in the trade-off between *precision* and *tractability*: abstraction replaces the potentially infinite or intractable concrete semantics with a simplified representation. This replacement, however, inevitably entails a **loss of information**: distinct concrete states may be mapped to the same abstract element, and concrete computations may therefore be

indistinguishable in the abstract domain. Due to this mismatch, the results computed in the abstract domain do not always align with those in the concrete domain. For example, if an abstract element represents all integers between 0 and 10, then the concrete operation "add 1" produces different results depending on whether the input was 0, 5, or 10. The abstract operation, instead, must approximate all these outcomes simultaneously, typically over-approximating them. This discrepancy raises two fundamental requirements:

(1) **Soundness**: every property that holds in the concrete domain must also be reflected in the abstract domain. This ensures that no behavior is lost by abstraction.
(2) **Completeness**: ideally, the abstraction should not introduce spurious results, so that the abstract and concrete computations coincide. Since this is often too strong a requirement in practice, completeness is usually studied as an additional property that may or may not hold.

In most applications, soundness is non-negotiable, while completeness can be relaxed. In what follows, we provide the formal definitions of these two notions.

Soundness. In abstract interpretation, soundness of abstractions can be expressed in two equivalent ways [CC77].

Definition 2.8 (Sound abstraction)**.** *Let $\langle A, \alpha, \gamma, C\rangle$ be a GI, let $f : C \to C$ be a function on the concrete domain, and let $f^\sharp : A \to A$ be its abstraction. We say that the abstraction is* sound *if*

$$\forall x \in C.\ \alpha \circ f(x) \leq_A f^\sharp \circ \alpha(x)$$

or, equivalently,

$$\forall x \in C.\ f \circ \gamma(x) \leq_C \gamma \circ f^\sharp(x).$$

This means that the abstract domain safely describes everything that is captured by the concrete semantics. However, the abstraction may introduce some noise, including extra states in the abstract domain that are not related to the concrete execution.

Figure 2.1 provides a graphical representation of this principle. In particular:

- Figure 2.1a illustrates the condition $\alpha \circ f(x) \leq_A f^\sharp \circ \alpha(x)$, comparing computations entirely in the abstract domain.
- Figure 2.1b illustrates the equivalent condition $f \circ \gamma(x) \leq_C \gamma \circ f^\sharp(x)$, comparing computations in the concrete domain.

It is worth noting that there always exists a particular abstract function $f^\sharp$ that guarantees soundness for f.

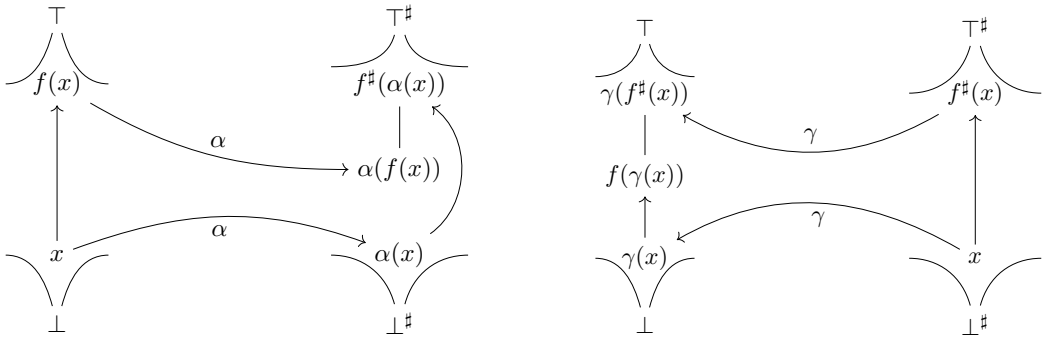


Figure 2.1: Soundness conditions, via α(a) and via γ(b)

Theorem 2.1.

$$\forall x \in C.\ \alpha \circ f(x) \leq_A f^\sharp \circ \alpha(x) \iff \forall x \in C.\ \alpha \circ f \circ \gamma(x) \leq_A f^\sharp(x).$$

As a direct consequence, we obtain that $\alpha \circ f \circ \gamma : A \to A$ represents the *best correct approximation (bca)* of $f : C \to C$ within the abstract domain [CC77].

Completeness. We now turn to the dual notion of *completeness*. While soundness ensures that abstract computations safely over-approximate concrete ones, completeness investigates under which conditions abstract and concrete computations yield results with the same precision.

As in the case of soundness, one can compare computational processes in either the abstract or the concrete domain. However, unlike soundness, the two characterizations of completeness are not equivalent: requiring equality in both settings leads to different notions.

Definition 2.9 (Complete abstraction)**.** *Let $\langle A, \alpha, \gamma, C\rangle$ be a GI, and let $f : C \to C$ be a concrete function. We say that $f^\sharp : A \to A$ is* complete for *f on the abstract domain $\alpha(C)$ if*

$$\alpha \circ f = f^\sharp \circ \alpha.$$

This condition requires that computations in the abstract domain preserve exactly the precision of the concrete semantics, i.e., the abstract and concrete results coincide. A graphical representation of completeness is provided in Figure 2.2.

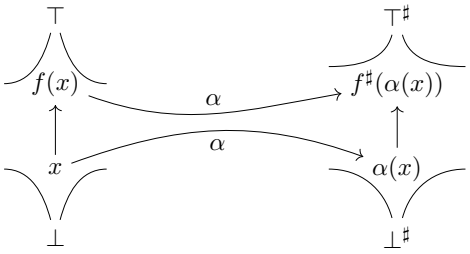


Figure 2.2: Completeness conditions

2.2 Quantum Computation

In this section, we review the mathematical foundations of quantum computation, emphasizing the concepts most relevant to the development of this thesis. For more details, we refer the reader to [NC10; Lin22; HZ08].

2.2.1 Linear Algebra for Quantum Mechanics

The standard formulation of quantum theory is based on the mathematical framework of *Hilbert spaces* and *linear operators*. Since the entire formalism of quantum computation relies on these structures, it is useful to recall the key notions of linear algebra that will be used throughout this thesis. We assume the reader has some familiarity with finite-dimensional inner product spaces.

A convenient and widely used formalism to represent vectors in quantum mechanics is the *Dirac notation*, also known as *bra-ket notation*. In this setting, a vector in a Hilbert space $\mathscr{H}$ is denoted by a *ket*,

$$|\psi\rangle \in \mathscr{H},$$

while the corresponding element of the dual space $\mathscr{H}^*$ is denoted by a *bra*,

$$\langle\psi| \in \mathscr{H}^*.$$

If $|\psi\rangle$ is represented as a column vector in $\mathbb{C}^d$, then its conjugate transpose is the bra $\langle\psi|$. Concretely,

$$|\psi\rangle = \begin{pmatrix} \psi_1 \\ \vdots \\ \psi_d \end{pmatrix}, \quad \langle\psi| = \begin{pmatrix} \psi_1^* & \cdots & \psi_d^* \end{pmatrix}.$$

The bra-ket notation is particularly well-suited to quantum theory, as it provides a compact and intuitive language for expressing inner products, tensor products, projectors, and

operator expressions.

Hilbert Space. Let $\mathscr{H}$ be a complex vector space. An *inner product* on $\mathscr{H}$ is a map

$$\langle \cdot | \cdot \rangle : \mathscr{H} \times \mathscr{H} \to \mathbb{C}$$

satisfying, for all $|\varphi\rangle, |\psi\rangle, |\phi\rangle \in \mathscr{H}$ and $c \in \mathbb{C}$:

(1) **Linearity in the second argument:** $\langle \varphi | c\psi + \phi \rangle = c \langle \varphi | \psi \rangle + \langle \varphi | \phi \rangle$.
(2) **Conjugate symmetry:** $\langle \varphi | \psi \rangle = \overline{\langle \psi | \varphi \rangle}$.
(3) **Positive definiteness:** $\langle \psi | \psi \rangle > 0$ whenever $|\psi\rangle \neq 0$.

The inner product is naturally represented as the pairing between a bra and a ket, $\langle \varphi | \psi \rangle = \langle \varphi | \psi \rangle$. It induces a norm

$$\|\psi\| = \sqrt{\langle \psi | \psi \rangle},$$

which measures the length of a state vector. A complex vector space equipped with such a structure is called an *inner product space*, often referred to in physics as a *scalar product space*.

Example 2.2. Let $\mathbb{C}^d$ be the space of all d-tuples of complex numbers. For two vectors $|\psi\rangle = (\psi_1, \ldots, \psi_d)$ and $|\phi\rangle = (\phi_1, \ldots, \phi_d)$, the standard inner product is

$$\langle \psi | \phi \rangle = \sum_{j=1}^{d} \psi_j^* \phi_j,$$

which is the convention universally adopted in quantum mechanics.

Two vectors $|\psi\rangle, |\phi\rangle \in \mathscr{H}$ are said to be *orthogonal* if $\langle \psi | \phi \rangle = 0$. A set of vectors is *orthogonal* if any two distinct vectors in it are orthogonal. If, in addition, each vector has norm one, the set is called *orthonormal*.

An inner product space is called *finite-dimensional* if it admits a finite basis; otherwise, it is infinite-dimensional. In finite dimensions, all norms are equivalent, and such spaces are automatically *complete* (all Cauchy sequences converge). In infinite dimensions, completeness is not guaranteed, and must be required explicitly:

Definition 2.10. *An inner product space that is complete with respect to the norm induced by the inner product is called a* Hilbert space.

Linear and Unitary Operators. Let $\mathscr{H}$ be a finite-dimensional Hilbert space over $\mathbb{C}$. A function $L : \mathscr{H} \to \mathscr{H}$ is called *linear* if for all $a_1, a_2 \in \mathbb{C}$ and $v_1, v_2 \in \mathscr{H}$ it satisfies

$$L(a_1 v_1 + a_2 v_2) = a_1 L(v_1) + a_2 L(v_2).$$

Given an orthonormal basis $\mathscr{B} = \{|e_1\rangle, \ldots, |e_d\rangle\}$ of $\mathscr{H}$, the action of L is completely determined by its values on the basis elements. Suppose $L(|e_j\rangle) = \sum_{i=1}^{d} a_{ij} |e_i\rangle$, $j = 1, \ldots, d$, where $a_{ij} \in \mathbb{C}$. Then L can be written in operator form as $L = \sum_{i,j=1}^{d} a_{ij} |e_i\rangle \langle e_j|$, and is represented in the basis $\mathscr{B}$ by the matrix $A = (a_{ij})_{i,j=1}^{d} \in \mathbb{C}^{d \times d}$. In this notation, the action of a linear operator L on a vector $|\psi\rangle$ is represented by the matrix multiplication $A|\psi\rangle$.

Example 2.3. Consider the case $d = 2$ with an orthonormal basis $\mathscr{B} = \{|0\rangle, |1\rangle\}$. Suppose L is defined by $L(|0\rangle) = |1\rangle$, $L(|1\rangle) = |0\rangle$. Then, in operator form, $L = |1\rangle\langle 0| + |0\rangle\langle 1|$. In the basis $\mathscr{B}$, the corresponding matrix is

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

which is precisely the Pauli-X operator.

A change of basis corresponds to a linear transformation. Let $\mathscr{B} = \{|e_1\rangle, \ldots, |e_d\rangle\}$ and $\mathscr{B}' = \{|\psi_1\rangle, \ldots, |\psi_d\rangle\}$ be two orthonormal bases of $\mathscr{H}$. Suppose each vector of $\mathscr{B}$ can be expressed as $|e_j\rangle = \sum_{i=1}^{d} b_{ij} |\psi_i\rangle, \qquad j = 1, \ldots, d$. Then the change-of-basis matrix is $B = (b_{ij})_{i,j=1}^{d} \in \mathbb{C}^{d \times d}$. If a vector has coordinates $x \in \mathbb{C}^d$ with respect to the basis $\mathscr{B}$, its coordinates in the basis $\mathscr{B}'$ are given by Bx.

Let $A \in \mathbb{C}^{d \times d}$. We recall the following operations:

- the transpose A^T, with $(A^T)_{ij} = A_{ji}$;
- the conjugate A^*, with $(A^*)_{ij} = (A_{ij})^*$;
- the adjoint $A^\dagger = (A^T)^*$.

A matrix $U \in \mathbb{C}^{d \times d}$ (or equivalently, the corresponding linear operator) is called *unitary* if

$$U^\dagger U = U U^\dagger = I,$$

that is the adjoint of the matrix correspond to is inverse.

Unitary operators are fundamental in quantum mechanics, since they correspond to reversible time evolutions and preserve the norm of state vectors. Equivalently, U is unitary if and

only if it preserves the inner product:

$$\langle Uu|Uv\rangle = \langle u|v\rangle \qquad \forall\, |u\rangle, |v\rangle \in \mathscr{H}.$$

2.2.2 Quantum States

In quantum theory, the states of finite systems are represented by unit vectors in finite-dimensional Hilbert spaces, most commonly $\mathbb{C}^{2^n}$ for an n-qubit register. The linear algebra introduced above provides the mathematical backbone for describing superposition, unitary evolution, measurement, and ultimately the entire framework of quantum computation.

The elementary carrier of information in quantum computing is the *qubit*, a vector in the two-dimensional Hilbert space $\mathscr{H} \cong \mathbb{C}^2$. The canonical orthonormal basis of this space consists of the two states:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

A generic qubit is a linear combination of these basis states,

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle, \qquad \alpha, \beta \in \mathbb{C},$$

which is referred to as a *superposition*. The complex amplitudes α and β determine the probabilities of observing the outcomes 0 or 1 upon measurement:

$$|\alpha|^2 + |\beta|^2 = 1,$$

with $|\alpha|^2$ (resp. $|\beta|^2$) being the probability of finding the system in state $|0\rangle$ (resp. $|1\rangle$).

2.2.3 Multi-Qubit Systems

Quantum algorithms rely on the interaction of multiple qubits. A composite system of m components with state spaces $\{\mathscr{H}_i\}_{i=0}^{m-1}$ is described by the tensor product space $\mathscr{H}_{\text{all}} = \bigotimes_{i=0}^{m-1} \mathscr{H}_i$, whose dimension is $\dim(\mathscr{H}_{\text{all}}) = \prod_{i=0}^{m-1} \dim(\mathscr{H}_i)$. If $\mathscr{B}_i = \{|e_j^i\rangle \mid j \in [0, N_i - 1]\}$ is an orthonormal basis of $\mathscr{H}_i$, then the tensor product space admits the orthonormal basis

$$\mathscr{B} = \left\{ |e_{j_0}^0\rangle \otimes \cdots \otimes \left|e_{j_{m-1}}^{m-1}\right\rangle \;\middle|\; j_i \in [0, N_i - 1] \;\; \forall i \right\}.$$

Thus a generic state in $\mathscr{H}_{\text{all}}$ can be expressed as $|\psi\rangle = \sum_{j_0,\ldots,j_{m-1}} \alpha_{j_0\cdots j_{m-1}} \left|e_{j_0}^0\right\rangle \otimes \cdots \otimes \left|e_{j_{m-1}}^{m-1}\right\rangle, \alpha_{j_0\cdots j_{m-1}} \in \mathbb{C}$. A particularly important case is the n-qubit system, with state space $\mathscr{H} = (\mathbb{C}^2)^{\otimes n} \cong \mathbb{C}^{2^n}$, which should be distinguished from $\mathbb{C}^{2n}$. We also adopt the notational conventions $|01\rangle \equiv |0\rangle \otimes |1\rangle$, $|0^{\otimes n}\rangle = |0\rangle^{\otimes n}$. For $x \in \{0,1\}^n$, we call x a *classical*

bit-string, and the set $\{|x\rangle \mid x \in \{0,1\}^n\}$ is referred to as the *computational basis* of $\mathbb{C}^{2^n}$.

2.2.4 Operators

An important family of unitary operators is given by the *Pauli matrices*:

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

These matrices form the generators of single-qubit quantum operations and will be frequently used in the sequel.

2.2.5 Quantum Operators

The evolution of a quantum state $|\psi\rangle \in \mathbb{C}^N$ is always governed by a unitary operator $U \in \mathbb{C}^{N\times N}$. In the setting of quantum computation, a unitary operator is often referred to as a *quantum gate*.

Example 2.4 (Single qubits Gates)**.** For a single qubit, the gates are linear operators on $\mathbb{C}^2$. Frequently used single-qubit gates include:

- The *Pauli matrices*:

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

- The *Hadamard gate*

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

  which maps $|0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, creating superposition.
- The *Phase gate*

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix};$$

- the *T gate*

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}.$$

Operators acting on multiple qubits are defined as tensor products of single-qubit operators, or as controlled gates. A controlled gate applies an operation to a *target qubit* only when its *control qubit(s)* are in state $|1\rangle$. For example, the two-qubit *controlled-NOT* (CNOT) gate acts as

$$\mathrm{CNOT}\,|y, x\rangle = |y, x \oplus y\rangle\,,$$

that is, it flips the target qubit x when the control qubit y is 1, and otherwise leaves it unchanged. CNOT plays a crucial role in generating entanglement between qubits, something unattainable with single-qubit operations alone.

Universality. Since a physical quantum computer can only implement a finite set of gates, arbitrary unitary operations must be approximated by compositions of gates from a fixed *universal gate set*. An example is the set $\{H, T, \text{CNOT}\}$, which can approximate any unitary operator on n qubits to arbitrary precision [Boy+00]. Another important family is the set of *Clifford gates*, generated by $\{H, S, \text{CNOT}\}$, which, while not universal, admits efficient classical simulation [Got98]

2.2.6 Measurement

Quantum measurement is an operation that allows us to extract a classical result from a quantum superposition $|\psi\rangle$. This operation transforms the quantum state into a classical one by breaking the quantum coherence (and so the quantum nature) of the state. Therefore, measurement is typically applied as the last operation in a quantum circuit to get the final (classical and probabilistic) result of the coherent (i.e., in superposition) evolution of the quantum system represented by the circuit. Formally, quantum measurement on the state space of the quantum system is represented using measurement operators $\{M_m\}$, where m corresponds to the possible outcomes of the measurement. If the system is in the quantum state $|\psi\rangle$ before the measurement, the probability of obtaining outcome m is given by $p(m) = \| M_m |\psi\rangle \|^2$, where $\| \cdot \|$ is the vector norm, and the system state after the measurement is $\frac{M_m|\psi\rangle}{\sqrt{p(m)}}$. For instance, given one qubit, the measurement operators are $M_0 = |0\rangle\langle 0|$ and $M_1 = |1\rangle\langle 1|$, corresponding to the outcomes 0 and 1. If the state of the qubit before the measurement is $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$, the probability of measuring 0 is: $p(0) = \|M_0\psi\|^2 = |\alpha|^2$ and the probability of measuring 1 is $p(1) = \|M_1\psi\|^2 = |\beta|^2$. After the measurement, if outcome 0 is observed, the state collapses to $\frac{M_0|\psi\rangle}{|\alpha|} = |0\rangle$ while if outcome 1 is observed, the state collapses to $\frac{M_1|\psi\rangle}{|\beta|} = |1\rangle$.

2.2.7 Quantum Circuit

Nearly all quantum algorithms operate on multi-qubit quantum systems. When quantum operators operate on two or more qubits, writing down quantum states in terms of their components quickly becomes cumbersome. The quantum circuit language offers a graphical and compact manner for writing down the procedure of applying a sequence of quantum operators to a quantum state. In the quantum circuit language, time flows from left to right, i.e., the input quantum state appears on the left, and the quantum operator appears on the right, and each "wire" represents a qubit. An example of a quantum circuit is shown in

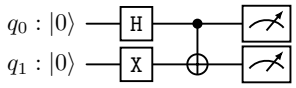


Figure 2.3: A simple quantum circuit implementing the unitary operator $\texttt{CX} \cdot (\texttt{X} \otimes \texttt{H})$ applied to $|00\rangle$.

Figure 2.3. In the circuit, the initial state 4 is $|0\rangle$ for both qubits. Two initial gates represent two particular unitary operators on one qubit, namely `H`, which transforms a basis vector into a superposition, and `X`, which is the unitary implementation of the classical NOT gate. All operations on 1 qubit correspond to rotations of the point in the unitary three-dimensional sphere (the so-called Bloch sphere) with polar coordinates corresponding to the qubit's amplitudes. The rotations, `Rx`, `Ry`, `Rz`, around the three axes x, y, z are universal for all operations on one qubit, in this case `H` is a rotation of a $\pi/2$ angle around the y axis while the NOT is a rotation of π around y or x. The next gate in the circuit is the controlled-NOT (`CX`) operation, which applies `X` to the second qubit only if the first qubit is $|1\rangle$. This circuit implements the unitary operator $\texttt{CX} \cdot (\texttt{X} \otimes \texttt{H})$. Here, the tensor product $(\texttt{X} \otimes \texttt{H})$ represents the operation `H` applied to the least significant qubit (q_0) and `X` to the most significant one, q_1. If the initial state is $|00\rangle$, where both q_0 and q_1 are $|0\rangle$, $|\psi_1\rangle = (\texttt{X} \otimes \texttt{H}) \cdot |00\rangle = |1\rangle \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, i.e. we have negated q_1 and applied `H` to q_0. The final state is $\texttt{CX}\,|\psi_1\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$. All quantum circuits can be constructed by using only gates in $\{\texttt{Rx}, \texttt{Ry}, \texttt{Rz}, \texttt{CX}\}$, i.e., this set is universal for quantum computation [Bar+95]. The final gate is the measurement operation, which is necessary to extract the final (classical) result from the quantum state $|\psi_2\rangle$ obtained from the quantum evolution of the input $|00\rangle$.

2.2.8 Quantum Variables

A *quantum variable* is a high-level abstraction of the state of a quantum register. Its type is the Hilbert space to which those states belong. In general, a variable associated with an n-qubit quantum register corresponds to a 2^n-dimensional Hilbert space. The abstraction of a qubit is a quantum variable q of type $\mathscr{H}_q \cong \mathbb{C}^2$. For quantum integer variables, instead, we consider the space of square-summable sequences $\mathscr{H}^\infty = \left\{ \sum_{n=-\infty}^{\infty} \alpha_n |n\rangle \;\middle|\; \alpha_n \in \mathbb{C},\ \forall n \in \mathbb{Z},\ \sum_{n=-\infty}^{\infty} |\alpha_n|^2 < \infty \right\}$, as discussed in [Rud87].

Following [Yin16], we define the space of values for a set of quantum variables $\mathbb{V} = \{q_i\}_i$ as the Hilbert space $\mathscr{H}_\mathbb{V} = \bigotimes_i \mathscr{H}_{q_i}$, obtained by composing the Hilbert spaces associated with each variable q_i via tensor product. Given a quantum program characterised by a set $\mathbb{V}$ of quantum variables, we call $\mathscr{H}_\mathbb{V}$ the *program space*, and the semantics of the program is given by vectors and operators in $\mathscr{H}_\mathbb{V}$.

We write $|\psi\rangle_{q_i}$ to indicate that the variable q_i represents the state $|\psi\rangle$ in $\mathscr{H}_{q_i}$. For entangled states, such as $\frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$, we use the notation $\left(\frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)\right)_{p,q}$ to denote that p and q correspond, respectively, to the first and second component of the entangled pair. In this case, the state is an inseparable vector in the space $\mathscr{H}_p \otimes \mathscr{H}_q$.

Given an operator U on $\mathscr{H}_{\mathbb{V}}$ and a subset $V \subseteq \mathbb{V}$, we write U_V to indicate that U acts on the variables in V and as the identity on all other variables in $\mathbb{V}$. For instance, if $\mathbb{V} = \{p, q, t\}$, the operator H_q denotes the Hadamard gate acting on (the type of) q and as the identity on the other variables. Its matrix representation is $\mathbb{I} \otimes H \otimes \mathbb{I}$, where $\mathbb{I}$ is the identity on $\mathbb{C}^2$ and H is the Hadamard matrix. Similarly, $CX_{p,q}$ denotes the controlled-NOT operator acting on p and q, and as the identity on the other variables.

2.2.9 Principles and Phenomena

The behaviour of quantum circuits is governed by the fundamental laws of quantum mechanics. As a result, quantum information exhibits phenomena with no classical counterparts. Unlike classical bits or even probability distributions over them, quantum states can display effects that are inherently quantum. These principles determine whether a quantum algorithm achieves its promised speedup in practice or fails to do so.

Entanglement

Entanglement can be intuitively described as a phenomenon in which the states of two or more quantum systems become strongly correlated. This occurs as a result of a computational process that produces a superposition of product states of the subsystems. The key feature is that the state of the composite system cannot be described independently of its components. For instance, consider a two-qubit state of the form $ab + cd$, where a, c are amplitudes associated with the possible states of qubit x and b, d with the possible states of qubit y. If x is measured in state a, then y must be in state b; similarly, if x collapses to state c, then y must be in d. As a concrete example, the Bell state:

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \ \in \ \mathscr{H}^2 \otimes \mathscr{H}^2$$

is entangled because it cannot be expressed as a tensor product of two single-qubit states. If one qubit is measured in state $|0\rangle$, the other immediately collapses to $|0\rangle$ as well, and analogously for $|1\rangle$.

Entanglement can also correlate a qubit with a superposition. For example, in the state

$$\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle),$$

if the first qubit is measured in $|0\rangle$, the second collapses to $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. If the first qubit is measured in $|1\rangle$, the second collapses to $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

Principle of Implicit Measurement

The *principle of implicit measurement* states that, without loss of generality, any qubits not explicitly measured at the end of a quantum computation may be assumed to be measured. This assumption does not affect the probability distribution of outcomes. Indeed, discarding a qubit—removing it from the computation—is equivalent to measuring it and ignoring the result.

Principle of Deferred Measurement

The *principle of deferred measurement* asserts that any measurement performed during a quantum computation can be postponed until the end. Intermediate measurements, together with classically controlled operations, can be replaced by fully quantum-controlled operations. For example, consider the circuits in Figure 2.4. In the first circuit (Figure 2.4b), the qubit q_0 is measured immediately after the Hadamard and CNOT gates. In the second circuit (Figure 2.4a), the measurement on q_0 is postponed, and its classical outcome is instead used to control the Pauli-X operation on q_1. Both circuits implement the same transformation, showing that measurements can be deferred until later in the computation without changing the final result.

(a) (b)

Figure 2.4: Two equivalent circuits showing the deferred measurement principle: a measurement on q_0 can be postponed and replaced with a classically controlled operation on q_1.

Principles of No-Cloning and No-Deleting

In classical computation, information can be freely copied and deleted. Quantum mechanics, however, imposes strict limitations on such operations.

No-Cloning. The *no-cloning theorem* [WZ82] states that no physical process can produce an exact copy of an arbitrary unknown quantum state. Formally, there exists no unitary operator that maps

$$|\psi\rangle \otimes |0\rangle \;\mapsto\; |\psi\rangle \otimes |\psi\rangle$$

for all states $|\psi\rangle$. Classical states $|0\rangle$ and $|1\rangle$ can be copied using the CNOT gate, but general superpositions cannot.

As an example, attempting to clone $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ using a CNOT gate yields

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle \xrightarrow{CNOT} \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle),$$

which is not $|+\rangle \otimes |+\rangle$ but the entangled Bell state.

No-Deleting. The *no-deleting theorem* [PB00] states that no unitary operator can erase an unknown quantum state. That is, no transformation exists that maps

$$|\psi_1\rangle \otimes |\psi_2\rangle \mapsto |\psi_1\rangle \otimes |0\rangle$$

for arbitrary $|\psi_2\rangle$. In the presence of entanglement, measurement may remove information, but at the cost of destroying correlations with the rest of the system.

Instead, temporary quantum data must be *uncomputed* [Ben73]: the sequence of operations is reversed to restore the variable to its initial value, so that it can be safely discarded.

As an example, the Bell state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ can be disentangled by applying the inverse of the CNOT gate (which is the CNOT itself), yielding

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \xrightarrow{CNOT} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle\,.$$

At this point, the second qubit is reset to $|0\rangle$ and can be safely discarded, while the first retains the superposition.

Chapter 3

Quantum Programs Analysis

As in the classical case, analysing a quantum program requires different approaches depending on the property we want to analyse. Specifically, one can be interested in properties of the quantum program variables, which are detectable by statically analyzing the flow of execution within the program. Classically, this is the case of the typical dataflow analyses used for compiler optimization (e.g., constant propagation, reaching definitions, copy elimination, register allocation, etc.), which aim to check whether variables are properly used, to verify that operations are applied in the correct order, and to ensure that no unintended duplication or discarding of variables occurs. The methodology in this case remains essentially the same as in the classical setting and consists of defining algorithms based on the execution flow structure of the program to extract information about variable properties. The key difference lies in the properties of interest, which are shaped by the features of quantum computation and are not shared by the classical paradigm. For example, duplication or discarding of variables is non-trivial in quantum languages, as it conflicts with principles such as the no-cloning theorem [NC10, Chapter 12] and implicit measurement [NC10, Chapter 4]. It is therefore essential to develop analyses that can prevent or detect such incorrect behaviours. While the infrastructure of these analyses is classical, their necessity stems from quantum-specific constraints. In this chapter, we refer to such approaches as *Flow-Based Analyses*.

On the other hand, one can also be interested in properties of the computational states, which are inherently quantum, such as entanglement. In this case, classical techniques cannot be directly reused, since reasoning requires the construction of abstract domains specifically designed to capture such phenomena. The difficulty is amplified by the exponential growth

of the Hilbert space with the number of qubits, which makes exact state representations infeasible. Abstract interpretation [CC77] offers a suitable framework to address this issue: abstract domains can approximate quantum states, enabling program reasoning or abstract simulation with a controlled degree of precision. Designing such approaches, which we refer to as *Domain-Based Analyses*, is therefore a central challenge in quantum program analysis.

In this chapter, we first review the literature on flow-based analyses, highlighting how traditional techniques have been adapted to the quantum setting. We then turn to domain-based analyses, where abstract interpretation is employed to approximate quantum properties.

3.1 Flow-Based Analyses

These approaches are characterized by the application of classical techniques, such as dataflow analysis or type-checkers, to the case of quantum programs.

Pattern Based Approaches

Pattern-based analyses have been developed to detect potential bugs in a quantum program. They focus on identifying problematic patterns or inappropriate uses of variables that could lead to unexpected behaviours. An example is the QChecker tool by Zhao et al. [ZZM21; Zha+23], which analyses the abstract syntax tree (AST) of quantum programs in Qiskit, to detect common bug patterns. Inspired by the Bugs4Q dataset [Zha+21], QChecker identifies syntax-level issues such as the use of gates not supported by the backend, incorrect parameters in gate applications, misuse of measurements, inconsistencies between the number of qubits and registers, as well as various API and compilation errors. The tool focuses on local, syntax-driven problems, such as undefined or deprecated methods, invalid object calls, or redundant register declarations.

LintQ [PP23] takes a different approach by translating quantum programs into the CodeQL intermediate representation, thus allowing reasoning not only about syntax through the AST, but also about dataflow and control-flow. In this setting, the program is modeled as a database of facts encoding elements and their relationships, enabling both classical and quantum dataflow analyses. The latter is used to track the ordering of gate applications on qubits, a key aspect in detecting subtle bugs. Compared to other tools, LintQ recognizes the largest variety of patterns, drawing from both empirical studies and prior work. Moreover, it is the only approach that explicitly models complex circuits built from multiple sub-circuits, which allows the detection of additional errors such as incorrect usage of the composed API. LintQ also captures dataflow–related issues (e.g., applying gates to qubits after measurement, or measuring the same qubit multiple times), problems with resource allocation (such as

insufficient classical registers or unused qubits in a circuit), and violations of implicit API constraints (for example, composing circuits without using the composed result, or adding gates after transpilation).

Kaul et al.[KKB23] propose Quantum-CPG, an extension of Code Property Graphs[Yam+14] to the quantum setting. Their approach models both dataflow and control-flow of programs written in Qiskit and OpenQASM, and—similarly to LintQ—captures the execution order of quantum gates through a graph-based representation of quantum data flow. Beyond detecting programming errors, Quantum-CPG also identifies classical code smells in hybrid quantum-classical programs. A distinctive feature of this work is the explicit modelling of the connection between qubits and classical registers, which makes it possible to spot unused or redundant measurement results. For instance, the tool can detect when a measurement result bit is produced but never consumed, when result bits remain constant due to missing transformations in the quantum circuit, or when conditional statements depend on constant classical values. Moreover, it highlights superfluous quantum operations, such as gates that have no observable effect on measurements.

Ensuring Correctness

In quantum programming languages, variables represent information encoded in quantum states rather than in memory locations, as is the case for classical languages. While they offer similar abstraction benefits, quantum variables introduce unique challenges due to the physical principles they obey, i.e., the laws of quantum mechanics. Operations such as copying or assignment are of non-trivial realisation due to the no-cloning theorem, and unused quantum variables must be carefully handled to avoid implicit measurements that can affect entangled states.

High-level quantum languages such as QWIRE[PRZ17] and Guppy[Koc+] are equipped with linear typing systems ensuring that values are used exactly once, preventing unintended duplication or deletion of resources.

Silq[Bic+20] introduces a more fine-grained type system that checks whether a variable is in superposition (for instance, after the application of quantum gates such as the Hadamard) and reduces errors related to implicit discarding. In particular, it can automatically infer safe discards when no uncomputation is required, or when the discarded state does not affect the computation. However, this approach relies on annotations and is effective mostly in the presence of local variables and simple control flow. Similar to Silq, Qurts [HH25] extends the Rust type system to achieve lifetime information about variables to support automatic uncomputation.

Building on this line of work, ReQWIRE[Ran+18] introduces a series of syntactic checks to verify the correctness of user-defined uncomputation, based solely on circuit-level information.

In [ADM24b], we propose a static analysis approach to automate uncomputation in quantum programs. Unlike type system-based approaches, our analysis reduces programmer workload and minimizes error messages by relaxing the requirement for variables to be used "exactly once" and allowing them to be used at most once. The analysis is based on two dataflow analyses: one to check variable usage constraints and another to detect unused variables across execution paths. This information enables the compiler to automatically insert discard functions, increasing flexibility and allowing integration with other analyses for improved accuracy. We present in detail this analysis in Chapter 4.

Rovara et al. [RBW25] introduce a comprehensive debugging framework that integrates classical simulation with lightweight static analyses to assist developers in diagnosing faults in quantum programs. The framework employs a *cone-of-influence analysis* to identify the subset of instructions that may affect the qubits involved in a failed assertion, thereby reducing the search space for potential sources of error. Complementarily, an *interaction analysis* constructs an interaction graph for the qubits. This analysis is employed to check the entanglement assertion in the program. In fact, when two or more qubits appear in entanglement assertions without having interacted previously, it is a typical indication of misconfigured controlled operations.

Code Optimization

Static analysis is also a useful tool for optimizing quantum code, as it helps identify substitution patterns or eliminate redundant gates.

Peduri et al. [PBG22] introduce Quantum Static Single Assignment (QSSA), an SSA-based intermediate representation for quantum computing that positions itself between the works on optimization, as discussed in this section, and those on correctness checks, presented in the previous one. They formalize the semantics of QSSA for hybrid quantum–classical programs and propose a static no-cloning verification algorithm, quadratic on general control-flow graphs and linear on structured ones. Classical SSA analyses and transformations are adapted to preserve quantum invariants, enabling both redundancy elimination and peephole optimizations (e.g., CNOT–CNOT elimination, unitary merging, and Pauli/Hadamard simplifications). QSSA can also leverage standard optimizations such as loop unrolling and dead-code elimination.

Ittah et al. [Itt+21] present QIRO, a quantum intermediate representation based on MLIR and SSA, designed to explicitly model the dataflow of quantum states. QIRO represents

the flow of quantum states between gates, allowing the reuse of both classical optimization techniques (e.g., constant folding, common subexpression elimination, inlining, loop unrolling, and dead-code elimination) and quantum-specific ones (e.g., unitary gate cancellation, rotation merging, and loop-boundary optimizations). The SSA structure enables optimization of side-effect-free quantum operations in the same way as classical ones, while preserving quantum invariants. QIRO also supports partial lowering and decomposition of high-level quantum operations, as well as resource estimation by lowering gates to classical counters.

Chen et al. in [CMS25] introduce a circuit optimization technique that exploits context information from the classical part of hybrid quantum algorithms. The central observation is that many measurement outcomes in hybrid workflows are never used in the subsequent classical computation. By formally distinguishing between valid qubits (those whose measurement outcomes are actually consumed) and dead qubits (those whose outcomes are discarded), they design an elimination algorithm that iteratively scans the circuit and removes dead gates without altering the probability distribution of valid outcomes. This approach, inspired by classical liveness analysis, adapts the idea to the setting of hybrid quantum–classical computation.

Behler et al. [Beh+25] present QStatic, a tool for static analysis and refactoring quantum programs. The work exploits the srcML[CDM11; CDM13] infrastructure to support both classical and quantum constructs in Abstract Syntax Trees (ASTs). QStatic implements automated refactoring rules for common quantum patterns, including iteration restructuring, Hadamard gate reduction, and code encapsulation into custom gates, improving both code readability and efficiency.

Resource Estimation

A static analysis of quantum code can also be useful for estimating the resources required by a quantum program or circuit.

ScaffCC [Jav+14] introduces timing and resource analysis. Resource analysis estimates the number of qubits and gates required, providing early feedback on algorithm efficiency before execution on hardware. Timing analysis estimates the circuit's critical path length under the assumption of unbounded resources, respecting quantum data dependencies enforced by the no-cloning theorem.

Colledan and Dal Lago [CD25] introduce a flexible type-based approach for quantum resource estimation in Quipper [Gre+13b]. They define a family of type systems, Proto-Quipper-RA, capable of deriving upper bounds on circuit size according to various metrics (e.g., width, depth, gate count). The system combines refinement types for local wire-level

metrics and effects for global circuit metrics, using arithmetic index terms to express bounds parametrically. The framework is provably correct with respect to the chosen metric and is implemented in the QuRA tool.

3.2 Domain-Based Analyses

We now turn our attention to the problem of analysing properties of quantum states. Broadly speaking, we can classify the approaches to this problem into three categories. The first category targets a specific property of quantum computation, namely entanglement, and develops abstract domains. The second category introduces general-purpose abstractions of quantum computation, aiming to represent quantum states (or families of states) efficiently, without explicitly dealing with their exponentially growing size. Finally, the third category exploits abstraction to optimize quantum circuits.

Entanglement Analyses

These works focus on providing a sound representation of entanglement in quantum programs. Their goal is to determine, at a given program point, whether two variables are entangled or not. To ensure soundness, these approaches overapproximate the entanglement property: if the analysis reports that two variables are not entangled, this is guaranteed to be correct; however, some variables may be conservatively considered entangled even if they are not.

ScaffCC [Jav+14] introduced the first static analysis of entanglement. The approach assumes that any two-qubit gate may create entanglement and records pairs of control and target qubits, together with a timestamp. When the same gate is encountered again, the analysis checks whether control qubits have changed state in the meantime (i.e., served as targets of other gates): if not, the second gate is recognized as the inverse of the first, and disentanglement is recorded; otherwise, both entanglements are preserved. As a result, the tool can identify when ancilla qubits remain entangled at the end of a module, issuing a *disentangled qubit check* warning whenever qubits are not properly uncomputed or measured, which could otherwise lead to incorrect outputs.

In [Per08b], Perdrix introduces the first entanglement analysis based on abstract interpretation. The analysis provides a sound approximation of the property. The domain combines state value information and entanglement. It is composed of a map from the qubit to one of four values: $\bot$ (both bases), s (standard), d (diagonal), or $\top$ (none), where s states that the qubit is in a standard basis, d that it is in a diagonal basis. Entanglement is represented by a partition of qubits: qubits in the same set may be entangled, and qubits in different sets are separable. The target is a small imperative language with qubits as memory, supporting

sequential composition, conditionals, loops, and a universal set of gates (Pauli, Hadamard, Phase, CNot). Conditional and loop constructs are typically treated by using measurements. Single-qubit gates update basis info, while two-qubit gates may merge blocks to approximate entanglement. Measurements separate the measured qubit and fix it in the standard basis; loops are handled via least fixed points. This approach is efficient, but it lacks the ability to detect when entanglement is nullified.

Honda [Hon15] extends the stabiliser formalism to handle the uncertainty introduced by non-Clifford gates such as the T gate. The key idea is that the effect of a non-Clifford gate can be approximated locally and, when possible, neutralised later in the computation. In general, abstract analyses of quantum programs use stabiliser-based representations[AG04] to track entanglement and basis information. When a non-Clifford gate is applied, the exact stabiliser information is lost, but the effect can be conservatively approximated, allowing the analysis to continue soundly while retaining as much information as possible. Honda's work targets a similar language to Perdrix's previous work. Conditional and Loop are performed via join operations, which combine abstract states from different branches, ensuring a safe approximation of entanglement evolution. This approach offers a more precise method for tracking entanglement, surpassing the previous one. However, it dealt with non-stabilizer states simply by treating them as unknown, which could propagate imprecision throughout the program. This could be useful in a few cases, such as when a non-stabilizer state is quickly discarded; however, it generally means the system cannot meaningfully discuss non-Clifford circuits.

In [Ran+20; Ran+21], Rand et al. introduce a type system based on Gottesman's [Got98] representation of Clifford gates (H,S,CX). The key idea is to type states during simulation—represented using stabilizers—so that separability can be checked efficiently. This approach proposes an entanglement analysis, although working at the circuit level (i.e., no control flow) and limited to Clifford gates and measurements.

The design of the language Twist [YMC22] includes the verification of the separability of quantum states, combining a type system with manual annotations and runtime dynamic checking based on classical simulation. Twist is a functional quantum language including classical Booleans, pairs, functions, qubits, measurement `let` and `if`-expressions. The key idea of Twist is *pure expression*, i.e., an expression whose evaluation is unaffected by the measurement outcomes of any qubit that does not belong to it, ensuring that its qubits remain separable from the rest of the program. Twist introduces a sound purity type system that conservatively tracks entanglement, purity assertion operators, which let programmers explicitly assert the separability of expressions or components of entangled pairs, and a hybrid verification approach, where static analysis is used together with runtime checks

based on Schmidt decomposition. In Twist, the static analysis is based on a type system, and it performs simple checks. Its precision is similar to the one introduced in [Per08b]. The dynamic check is instead used mainly to check assertions where the entanglement is nullified during the computation.

In [XZ23], Xia et al. present the first static entanglement analysis method for the quantum programming language Q# [Svo+18]. The analysis is built on an interprocedural control flow graph and tracks the operations applied to each variable in order to determine whether it represents a classical state or a superposition. By doing so, it identifies both the creation and the cancellation of entanglement. The approach is illustrated, focusing on Hadamard, single-qubit rotation, and CNOT operations.

Finally, our work [ADM24a; ADM25] introduces a static analysis based on a new abstract domain for entanglement. This domain not only determines the sets of entangled variables, but also distinguishes a particular relation, which we call *direct inseparability*, corresponding to entangled states of the form $\alpha\,|00\ldots0\rangle + \beta\,|11\ldots1\rangle$. Moreover, the domain incorporates additional labels that abstract the quantum states of variables. These labels enable a static analysis that not only identifies entangled variables but also approximates their quantum state. Our approach improves the accuracy of existing methods that rely on a more imprecise domain, such as the analysis in [Per08b], while targeting a basic imperative quantum language equipped with a universal gate set (CNOT, Hadamard, and Phase gates) and measurement. We will discuss the details of our analysis in Chapter 5. An ML implementation of our domain proposed in [ADM24a] is presented in [RMN25].

Abstracting the Quantum State

The approaches described here do not introduce an abstract domain with the goal of capturing a specific property (such as entanglement, as discussed before). Instead, they focus on representing the overall state of the computation.

Yu et al. [YP21] introduce a method, called Quantum Abstract Interpretation (QAI), to simplify the analysis of quantum programs by reducing the dimensionality of the quantum state. Instead of considering all n qubits at once, their approach partitions the system into smaller groups of qubits, each represented by a lower-dimensional abstraction. This allows for efficient static verification of certain properties and assertions in quantum circuits without simulating the entire state. The approach uses mathematical operations to combine and compare these smaller abstractions, ensuring that the results still soundly approximate the behavior of the full system. It can check whether specific states yield in a subspace, described as a span of standard bases. However, the abstraction loses precision because it breaks global correlations among qubits into simpler relations between subsets of them. Building on

this abstraction, recent work [YPR25] introduces the logic framework *SAQR-QC*. SAQR-QC combines *Quantum Hoare Logic* [Yin12] with the principles of QAI. In particular, it allows for performing QHL-style reasoning about program behaviour, using QAI approximation to ensure scalability.

In [Che+23b; Che+23a; Abd+25; Che+25], the authors introduce a verification methodology for quantum circuits and programs based primarily on tree automata. Tree automata (TAs) provide a formal model for representing sets of trees and, in the context of quantum verification, are used to compactly encode sets of quantum states. In the original AutoQ framework [Che+23b], TAs were employed to efficiently represent pre- and post-conditions of quantum circuits. The semantics of quantum gates are implemented in the TA domain, and verification is reduced to checking whether the output-state TA is included in the post-condition TA. In [Che+23a], the framework was extended to use symbolic TAs, introducing symbolic variables in the leaves of the automata to enable reasoning about relational properties between quantum states. In [Abd+25], the authors introduced Level-Synchronized Tree Automata (LSTAs) to offer a more compact symbolic representation that allows verification of quantum circuits with at most quadratic complexity per gate. Moreover, LSTAs support parameterized verification, making it possible to analyze entire families of circuits with arbitrary numbers of qubits. Finally, AutoQ 2.0 [Che+25] generalizes the framework from circuits to full quantum programs, incorporating classical control structures such as branches and loops, while retaining the symbolic and scalable verification capabilities enabled by TAs and LSTAs.

The path-sum representation, introduced in [Amy19], provides a symbolic and compositional representation of quantum circuits, expressing their behavior as a sum over all computational paths. Formally, a circuit acting on input variables x can be represented as $|x\rangle \mapsto \frac{1}{\sqrt{2^m}} \sum_{y \in 0,1^m} e^{i\pi P(x,y)} |f(x,y)\rangle$, where y enumerates the path variables, $P(x,y)$ encodes the phase accumulated along each path, and $f(x,y)$ describes the corresponding output transformation. This representation enables compositional reasoning, allowing equivalence proofs of large circuits to be constructed modularly, without explicitly manipulating exponentially large state vectors or matrices. Path-sums have been effectively employed for the automated equivalence checking of Clifford+T circuits. However, the original formalism is limited to fixed-size circuits and cannot directly capture parametrized circuit-generating programs. The Qbricks tool [Cha+21] extends the path-sum semantics to support verification of parametric quantum programs, i.e. quantum programs with a parametric number of qubits.

Hietala et al. [Hie+21] introduce *VOQC*, the first fully verified optimizer for quantum circuits. Their work relies on a symbolic matrix semantics that avoids the exponential

blow-up of concrete $2^n \times 2^n$ matrices by manipulating algebraic expressions denoting the underlying linear operators. This approach enables correctness proofs for optimizations that apply to circuits with a parametric number of qubits. Circuit rewrites are formalized as equivalence proofs between symbolic matrix expressions, supported by Coq tactics such as `gridify` [Nas+25] for normalizing tensor products and matrix multiplications.

Bichsel et al. [Bic+23] present *Abstraqt*, an abstract stabilizer simulator for efficiently analysing quantum circuits, including those with non-Clifford gates. It overcomes the exponential growth of summands in exact simulation by using abstract interpretation to compress multiple concrete states into a single abstract representation, trading some precision for efficiency. Quantum states are represented with Pauli operators and complex intervals, and Abstraqt provides abstract domains and transformers to handle gates and measurements, enabling static analysis and verification of circuits that are infeasible to simulate exactly.

Our work [AMM25] applies abstract interpretation to analyze Variational Quantum Circuits (VQCs). VQCs are hybrid quantum-classical models, characterized by a set of trainable parameters that are optimized during the learning process, similar to neural networks. In this work, we extend the classical interval abstract domain [CC79; MKC09] to the quantum setting, representing the quantum state as a vector of intervals. This representation allows them to model noisy inputs to the VQC and analyze the circuit's robustness against noise and adversarial perturbations. We will discuss the details of this work in Chapter 6.

Abstraction for Optimization

Finally, we review a set of works where abstraction is employed to reduce simulation complexity or to optimise quantum programs. Unlike the approaches discussed in Section 3.1, which primarily rely on dataflow information (such as liveness), these methods exploit semantic properties—namely, approximations of the program state—to guide the optimisation process.

In [LBZ21], Liu et al. introduce the Relaxed Peephole Optimization (RPO). This approach tries to statically determine the state of the qubits to reduce the number of gates. They use labels to indicate that a qubit is in one of the X-, Y-, or Z-basis states together with a $\top$ label and compute the approximation of the execution on these labels. Based on this information, they replace expensive operations with equivalent but less costly ones.

In [CS23], Chen et al. introduce an optimization technique, called Quantum Constant Propagation, that exploits partial circuit execution and removes superfluous controlled gates, leveraging the common assumption that all qubits are initially in the $|0\rangle$ state. The key idea is to keep non-entangled groups of qubits separated for as long as possible and to describe

their state in a compact form. To avoid excessive complexity, a bound is imposed on the size of the entangled state that can be represented; once this bound is exceeded, the state of the group is abstracted into a form that signals a loss of information. In [CFM25], based on the description of mid-circuit measurement presented in [CFM24], the Quantum Constant Propagation is extended to such circuits.

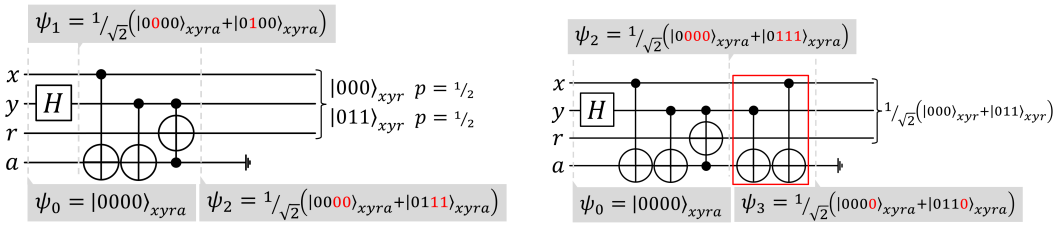
Amy et al. [AL25] generalise the phase folding optimisation [AMM14], traditionally used to reduce costly T-gates in straight-line quantum circuits, so that it can also be applied to quantum programs with classical control flow. The key idea is to reinterpret phase folding as a form of relational analysis, in particular, affine relation analysis. They also incorporate a non-linear analysis, which makes it possible to optimise programs involving operations such as Toffoli gates, whose effect corresponds to non-linear transformations on classical states. To address precision losses caused by interference when composing transition relations, the authors introduce a sum-over-paths technique [Amy19]. This work also shows that classical program analysis can be used on classical data in superposition, making classical techniques feasible for quantum programs.

Chapter 4

Static Analysis of Quantum Programs

Almost all high-level programming languages provide variables as information holders. However, while in classical languages, variables are abstractions of memory regions where information can be stored and retrieved, in quantum languages, they are rather abstractions of the information contained in the physical space of quantum states. Although the advantages offered by these abstractions (in terms of easy writing for the programmer) are the same in both classical and quantum contexts, quantum variables introduce new challenges strictly related to the specific features of quantum computation. In fact, the implementation of a simple operation such as duplicating a quantum variable is not trivial due to the *no-cloning* theorem [NC10, Chapter 12]. It is also very important (contrary to the classical setting) to take care of unused quantum variables, which, if not appropriately dealt with, could have a negative impact on the correctness of the result of the whole program due to the *implicit measurement* principle [NC10, Chapter 4]).

Entanglement is a powerful feature of quantum computation; it is at the base of important quantum communication protocols, which cannot be realized by classical means (see, for example, quantum teleportation [NC10, Chapter 1.3]). However, this feature introduces some complications when it comes to the implementation of quantum programming languages due to the inevitable presence of temporary variables in a program. At the circuit level, the problem arises in the computation of a classical irreversible function on a quantum computer, as this requires additional (auxiliary) qubits to obtain reversibility and then unitarity[KSV02, Chapter 7]. It turns out that when feeding the circuit with a superposition of input states, the auxiliary qubits become entangled with the output qubits, and their elimination (which induces an implicit measurement) affects the state of the output qubits. The solution to



(a) (b)

Figure 4.1: A circuit computing $(x \oplus y) \wedge y$ without (a) and with (b) uncomputation of a.

this problem is to 'uncompute' these auxiliary qubits before their elimination, returning them to their initial unentangled state. Therefore, at the program level, temporary quantum variables cannot easily be dropped as we do with temporary values in classical programs, and a careful compilation is necessary to avoid the generation of an incorrect circuit. Moreover, quantum variables cannot be overwritten because this would imply copying a quantum state at the physical level, which, as we know, is impossible [JHG19, Chapter 5].

We illustrate the problem of uncomputation with the circuit in Figure 4.1a. This circuit computes $(x \oplus y) \wedge y$, i.e., the conjunction of y and the exclusive-or between x and y, using a temporary qubit a and storing the result in r. Firstly, we apply the Hadamard operation (H) to y, thus obtaining the state $\psi_1 = {}^1/\sqrt{2}(|0000\rangle_{xyra} + |0100\rangle_{xyra})$; secondly, we compute $x \oplus y$ through the two controlled-NOT gates obtaining the state $\psi_1' = {}^1/\sqrt{2}(|0000\rangle_{xyra} + |0101\rangle_{xyra})$; Finally, we calculate $a \wedge y$ in r using a Toffoli gate* getting the state $\psi_2 = {}^1/\sqrt{2}(|0000\rangle_{xyra} + |0111\rangle_{xyra})$. Now, if we decided to delete the qubit a (since it is no longer needed), the state of the first three qubits would collapse with probability ${}^1/_2$ to $|000\rangle_{xyr}$ or $|011\rangle_{xyr}$, depending on the result of the implicit measurement. This incorrect result is due to the principle of implicit measurement [NC10, Chapter 4] (deleting a qubit means measuring it) and the entanglement introduced by the controlled-NOT gate.

The example clearly shows the need of uncomputing all temporary variables by resetting their initial state to ensure their (implicit or explicit) removal without side effects. The simplest way to uncompute a qubit is to take all previously applied operations and reapply their inverses in reverse order, returning the qubit to its initial unentangled state. In Figure 4.1b, we show that if we uncompute a before the final measurement, introducing additional gates to bring the value of a back to $|0\rangle$ before deleting it (causing an implicit measurement), we do not lose information about the other three qubits.

As we have just shown, unused quantum variables correspond to portions of the quantum circuit that would be measured even if no explicit measurement operator is applied, and

* The Toffoli gate corresponds to a double controlled NOT: the target is negated if both controllers are 1.

```
def used_consume(qubit: a):
    b = h(a)
    c = t(a) # Error, a is not defined
    return b,c
```

Figure 4.2: Simple function that uses a consumed variable

```
def xor_and(x: qubit,y: qubit):
    a,r = qubit(), qubit()
    x,a = cx(x,a)
    y,a = cx(y,a)
    a,y,r = toff(a,y,r)
    # Error: a is not 'consumed'

    return x,y,r
```

(a) Guppy program corresponding to the circuit in Figure 4.1a

```
def xor_and(x: qubit,y: qubit):
    a,r = qubit(), qubit()
    x,a = cx(x,a)
    y,a = cx(y,a)
    a,y,r = toff(a,y,r)
    discard(a)

    return x,y,r
```

(b) Guppy program with safe variable discarding

Figure 4.3

this may have unexpected side effects when it involves entangled states. Thus, a quantum program with unused quantum variables requires careful implementation, ensuring that at the circuit level, they are appropriately 'reset' before the end of the execution of the program to eliminate the presence of entanglement. This process is commonly referred to as uncomputation. Thus, avoiding the copy and the implicit discarding of quantum variables is crucial for the correctness of the program results.

A typically adopted solution is linear typing, which forces the programmer to use variables exactly once so that when a variable x is used the first time, it is 'consumed' and no longer available. We will exemplify the effects of a linear type system by means of programs written in the Guppy language [Koc+], a Python-embedded language, which we take as a model of a quantum language with linear typing. A program like the one in Figure 4.2 does not pass the check of a linear type checker since the variable `a` is used twice.

Linear typing also ensures that no unused variables occur in a program, i.e., all program variables must be consumed before the end of the execution. For example, consider the implementation of the circuits in Figure 4.1 in Guppy, given in Figure 4.3. The Guppy compiler would reject the program in Figure 4.3a since `a` is not consumed. Instead, the code in Figure 4.3b would be assessed as well-typed since the Guppy primitive `discard`(a) consumes `a`. In general, we cannot say if this program will be compiled in the circuit in Figure 4.1b since it depends on the implementation of the `discard` primitive; nevertheless, the final result, after the return, will always be the same correct result. Implicit discarding also occurs when we redefine a non-consumed variable, as in the code of Figure 4.4a. This program is ill-typed since the variable `b` is not consumed when redefined. In this case, the

```
b = h(a)
# error: b not consumed
b = qubit()
```

(a) Wrong re-definition

```
b = h(a)
discard(b) #'drop' b
b = qubit()
```

(b) Proper re-definition

Figure 4.4: Two simple programs where we assume that `a` is already defined

type checker returns an error indicating that the first `b` is not consumed. Instead, the code in Figure 4.4b is well-typed since we properly discard `b` before redefining it.

With the aim of relaxing the constraints imposed by linear type systems, we propose a different approach, which is based on static analysis and consists of two steps: first, we check that variables are used *at most* once, filtering out programs that violate this rule; second, we identify unused and overwritten variables and automatically insert the discard function. Two key analyses are necessary for our approach: a forward data-flow analysis (Consuming Analysis) that collects information about the availability of variables at each program point and a backward data-flow analysis (Uncomputation Analysis) that collects information about the usage of variables. The information resulting from the first analysis allows the compiler to verify, in place of the type checker, that the programs use variables at most once. Then, the second analysis gives the necessary information on the program points where to insert the discard function, transforming a program that uses variables *at most once* into a program that uses variables *exactly once*. This procedure introduces greater flexibility than the type system approach, enhancing the language usability. In fact, with our analysis, we still reject programs that use consumed variables, but we automatically insert the `discard` when needed, relieving the programmer from this task. As an example, the function in Figure 4.2 is still rejected, but functions in Figure 4.3a and Figure 4.4a are automatically transformed at compile time, respectively, into the ones in Figure 4.3b and Figure 4.4b.

4.1 A Control Flow Graph Language

Static analysis is usually performed using the control flow graph (CFG) representation of programs [SWH12]. We define our static analyses as data-flow analyses based on control flow graphs. Following [OHe19; Bru+23], we consider the Control Flow Graphs (CFG) language defined by the syntax:

$$\begin{aligned} \mathtt{c} &::= \mathsf{label} \mid \mathtt{c}; \mathtt{c} \mid \mathtt{c} \oplus \mathtt{c} \mid \mathtt{c}^* \\ \mathsf{label} &::= \mathtt{NonZero(b)} \mid \mathtt{Zero(b)} \mid \mathsf{stm} \end{aligned} \tag{4.1}$$

(a) label (b) $\mathtt{c}_1;\mathtt{c}_2$ (c) $\mathtt{c}_1 \oplus \mathtt{c}_2$ (d) $\mathtt{c}^*$

Figure 4.5: Graphical representation of language $\mathtt{c}$, the CFG.

where the term $\mathtt{c};\mathtt{c}$ represents sequential composition; the term $\mathtt{c} \oplus \mathtt{c}$ is the choice command that corresponds to the execution of one of the two possible branches; the term $\mathtt{c}^*$ is the Kleene closure of $\mathtt{c}^n$, $n \in \mathbb{N}$, where $\mathtt{c}^n$ is the composition $\mathtt{c};\ldots\mathtt{c};$, n times. For a Boolean expression $\mathtt{b}$ and a statement stm (both defined by the grammar of a specific language), $\mathtt{NonZero}(\mathtt{b})$ is a special label that indicates that in its path $\mathtt{b}$ is true. In contrast, $\mathtt{Zero}(\mathtt{b})$ indicates that the Boolean condition $\mathtt{b}$ is false in its path. This syntax is general enough to cover deterministic imperative languages [Win93, Chapter 14, Exercise 14.4] whenever stm contains at least an assignment statement and a null operator, such as skip. In fact, in this language, we can write both the `while` and `ifthenelse` statements as follows:

$$\begin{aligned} \mathtt{if}\,\mathtt{b}\,\mathtt{then}\,\mathtt{c}_1\,\mathtt{else}\,\mathtt{c}_2 &\equiv (\mathtt{NonZero}(\mathtt{b});\mathtt{c}_1) \oplus (\mathtt{Zero}(\mathtt{b});\mathtt{c}_2) \\ \mathtt{while}\,\mathtt{b}\,\mathtt{do}\,\mathtt{c} &\equiv (\mathtt{NonZero}(\mathtt{b});\mathtt{c})^*;\mathtt{Zero}(\mathtt{b}) \end{aligned} \tag{4.2}$$

The language defined by (Equation 4.1) is interesting because it corresponds precisely to the control flow graph representation of programs, on which standard data-flow analysis is usually performed [SWH12]. The CFG associated with a program is a graph with a start node corresponding to the program entry point, an end node corresponding to the exit point, and all other nodes corresponding to intermediate points in the execution of the program; each edge of the graph has a label that represents the change produced by the execution of an instruction of the language that is analysed. Hence, label defines the language of CFG edge labels displayed in Figure 4.5a where $l \in$ label, while the other elements of $\mathtt{c}$ determine how we compose the edges depending on their labels as displayed in Figure 4.5b for the CFG corresponding to the sequential composition $\mathtt{c};\mathtt{c}$, in Figure 4.5c for the nondeterministic choice $\mathtt{c} \oplus \mathtt{c}$ and in Figure 4.5d for the iteration $\mathtt{c}^*$. In this way, we can define a new analysis simply by providing abstract semantics for the instructions defining stm and for the truth evaluation of $\mathtt{b}$, namely for the language of labels label.

In the following, if V is the set of program points, the CFG will be defined as sets of edges, labelled in label, between nodes in V, i.e., as subsets of $V \times \text{label} \times V$.

We will demonstrate our approach on the Guppy language [Koc+]. This is a Python-embedded language whose compiler runs within the Python interpreter, but the compiled

program is independent of the Python runtime. Guppy adopts Python's control flow (`if`, `for`, `while`, and `return` statements), allowing measurement outcomes as a guard.

When we analyse a concrete language, we need to specify the label category of the syntax in (4.1) by defining stm, i.e., the syntax of the language statements that will label the CFG. We can extend the label syntax to represent a control flow graph of the Guppy language. Like Python, a Guppy program is a set of function declarations. We can represent each function by a CFG and a program (i.e., a set of functions) as a set of CFGs.

Let $\mathbb{V}_q$ be the set of quantum variables, $\mathbb{V}_c$ the set of classical variables, and $\mathbb{V} = \mathbb{V}_q \cup \mathbb{V}_c$ the set of variables of a program. We use $\vec{x} \in \mathbb{V}_c^n$, $\vec{q} \in \mathbb{V}_q^n$, and $\vec{v} \in \mathbb{V}^n$, where $n \in \mathbb{N}$, for denoting, respectively, a list of classical, quantum, or both types of variables. Note that the list could be empty.

We denote by `b` a generic Boolean expression composed of classical variables or measurement of quantum variables (e.g. $\neg x \wedge \texttt{measure}(q)$) and `e` to indicate a generic classical expression (that does not contain any quantum variable). We can extend the syntax in Equation 4.1, with Guppy statements as follows:

$$\mathsf{stm} ::= \texttt{pass} \mid \texttt{A:}\ \vec{v} \mid \vec{v} = \mathsf{fun}(\vec{v}) \mid \vec{x} = \texttt{e} \mid \texttt{return}\ \vec{v} \mid \texttt{discard}(\vec{q}) \tag{4.3}$$

where `pass` is the Python statement that corresponds to `skip`, `A:` $\vec{v}$ declares which variables are the function arguments and fun indicates both built-in function (`measure`, quantum gates and initialization function `qubit`()) and user-defined functions.

4.2 Data-Flow Analyses

In this section, we introduce the two key analyses needed to implement our approach: a forward data flow analysis, which we call *consuming analysis*, gathering information about the availability of variables at each program point, and a backward data flow analysis, which we call *uncomputation analysis*, gathering information about the usage of variables.

Consuming Analysis

This analysis aims to detect the available variables at each node of the program's CFG, i.e., the variables that are defined and not yet consumed. To this aim, we must check that, in all paths, each used variable is defined and not yet consumed. This means that the analysis must be definite [SWH12], i.e., the information propagates among nodes by intersection.

Let $\mathbb{V}_q$ be the set of quantum variables. We define the lattice, $\langle \wp(\mathbb{V}_q), \supseteq \rangle$, of the powerset of $\mathbb{V}_q$ ordered by inverse inclusion. Thus, the top element is $\varnothing$, while the bottom is $\mathbb{V}_q$. For

expressions $\mathtt{e}$, we denote by $Q(\mathtt{e}) \subseteq \mathbb{V}_q$ the set of quantum variables in $\mathtt{e}$. We define the (abstract) consuming semantics $(\!|\, l \,|\!) : \wp(\mathbb{V}_q) \to \wp(\mathbb{V}_q)$ as the abstract edge effects defined for each label $l \in \mathsf{label}$, as follows:

$$\begin{aligned}
(\!|\, \texttt{NonZero(b)} \,|\!)\mathfrak{D} &= (\!|\, \texttt{Zero(b)} \,|\!)\mathfrak{D} = \mathfrak{D} \setminus Q(\mathtt{b}) \\
(\!|\, \texttt{pass} \,|\!)\mathfrak{D} &= (\!|\, \vec{x} = \mathtt{e} \,|\!)\mathfrak{D} = \mathfrak{D} \\
(\!|\, \texttt{return}\ \vec{v} \,|\!)\mathfrak{D} &= \mathfrak{D} \setminus Q(\vec{v}) \\
(\!|\, \vec{v}_1 = \mathsf{fun}(\vec{v}_2) \,|\!)\mathfrak{D} &= (\mathfrak{D} \setminus Q(\vec{v}_2)) \cup Q(\vec{v}_1) \\
(\!|\, \texttt{A:}\ \vec{v} \,|\!)\mathfrak{D} &= \mathfrak{D} \cup Q(\vec{v}) \\
(\!|\, \texttt{discard}(\vec{q}) \,|\!)\mathfrak{D} &= \mathfrak{D} \setminus \{\vec{q}\}
\end{aligned} \tag{4.4}$$

As a general rule, since any use consumes variables, the abstract semantics is simple: when a variable is used, it is removed from $\mathfrak{D}$, and it is added when defined. In particular, for Boolean expressions, since in $\mathtt{b}$ the only quantum variables that can be used are the ones that are measured, we remove $Q(\mathtt{b})$ from $\mathfrak{D}$.

Proposition 4.1 (Consuming semantics distributivity)**.** *The abstract semantics $(\!|\, \cdot \,|\!)$ defined in Equation 4.4 is distributive w.r.t. intersection, i.e., for all labels l and subsets $\mathbf{X} \subseteq \wp(\mathbb{V}_q)$, we have*

$$(\!|\, l \,|\!)(\cap \mathbf{X}) = \bigcap \{ (\!|\, l \,|\!)\mathfrak{D} | \mathfrak{D} \in \mathbf{X} \}$$

Sketch. This follows easily from the definition of $(\!|\, l \,|\!)$ and the properties of the set-theoretic operations (see e.g. [NNH99; Aho+06; SWH12]). □

To compute the CFG analysis, we need to compute the set $\mathfrak{D}$ for each node of the CFG [SWH12], namely at each program point of the analysed program. Since the analysis is forward, the set $\mathfrak{D}$ at node v, denoted $\mathfrak{D}[v]$, depends on the sets $\mathfrak{D}[u]$ of its predecessors u, and the label semantics of the edges entering v. Let **start** (**end**) be the starting (exit) node of a a CFG G. For each $v \in G$, we define

$$\mathfrak{D}[v] = \begin{cases} \varnothing & \text{if } v = \textbf{start} \\ \bigcap \{ (\!|\, l \,|\!)(\mathfrak{D}[u]) \mid (u, l, v) \in G \} & \text{otherwise} \end{cases}$$

thus obtaining a system of $n = |V|$ equations in n unknowns, which can be solved by fix-point, achieving the so-called Maximum Fixed Point (MFP) solution [SWH12]. As shown in [SWH12], this solution provides the best solution we can compute on a CFG. In fact, Proposition 4.1 implies that, for each $v \in V$, the computed available variables set $\mathfrak{D}[v]$

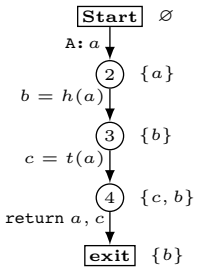


Figure 4.6: CFG of program in Figure 4.2 with the computed $\mathscr{D}$ for each node

corresponds to the MOP (Meet Over all Paths), namely

$$\mathscr{D}^*[v] = \bigcap \left\{ \; (\!|\, \pi \,|\!) \;\middle|\; \pi = k_1, \ldots, k_n \text{ is a path from } \textbf{start} \text{ to } v (\!|\, \pi \,|\!) \stackrel{\text{def}}{=} (\!|\, k_n \,|\!) \circ \cdots \circ (\!|\, k_1 \,|\!) \right\}$$

By using the equality with the MOP solution, we can prove that, for each program point, the computed set of available variables is indeed an under-approximation of the available variables; in other words, our analysis is sound.

Theorem 4.1. *Given an edge (u, l, v), if the command represented by l uses a variable q which has not been defined or has already been consumed in at least one path, then $q \notin \mathscr{D}[u]$.*

Sketch. This follows directly from how we combine the paths' semantics. In fact, if a variable is not defined or is consumed in at least one path, it will not be included in $\mathscr{D}[u]$ thanks to the intersection operator. □

As an example, consider the simple program in Figure 4.2 represented as CFG in Figure 4.6.

This analysis shows that in node 3, the variable a is not available; in fact, the edge $(3, 4)$ leads to an error.

Uncomputation Analysis

The uncomputation analysis determines the appropriate locations where to insert the discard function. To this purpose, we must identify those unused variables that lead to implicit discarding. For this analysis, the notion of *live* variable comes in handy. We recall that a variable x is called *live* at point u if u is in a path between a previous definition of x and a following use of x (without interleaving further definitions of x) [Aho+06; NNH99]. However, to figure out where it is necessary to uncompute, we need to know not only where a variable is live but also where the last use of the variable consumes it. Hence, we extend the notion of liveness to include this additional information.

Definition 4.1. *A quantum variable $q \in \mathbb{V}_q$ is* unsafe live *at point u if it is live and it is not consumed in at least one path starting from u; q is said to be* safe live *if it is consumed or returned in all paths from u.*

To simultaneously compute these two types of liveness, we define the abstract domain as a pair of sets of quantum variables. The analysis computes $(\mathcal{S},\ \mathcal{U}) \in \wp(\mathbb{V}_q) \times \wp(\mathbb{V}_q)$, where $\mathcal{U} \subseteq \mathbb{V}_q$ is the set of *unsafe* live variables and $\mathcal{S} \subseteq \mathbb{V}_q$ is the set of *safe* live variables. Note that while *unsafety* will be over-approximated as it requires that the property of being non-consumed holds at least in one exiting path, *safety* requires the property holding on all the exiting paths, leading therefore to an under-approximation. A computational ordering $\sqsubseteq$, allowing for simultaneously over-approximating *unsafe live* variables and under-approximating *safe live* variables, can be defined on the abstract domain $\wp(\mathbb{V}_q) \times \wp(\mathbb{V}_q)$ as follows.

Definition 4.2. *Let $(\mathcal{S}_1,\ \mathcal{U}_1), (\mathcal{S}_2,\ \mathcal{U}_2) \in \wp(\mathbb{V}_q) \times \wp(\mathbb{V}_q)$, we define*

$$(\mathcal{S}_1,\ \mathcal{U}_1) \sqsubseteq (\mathcal{S}_2,\ \mathcal{U}_2) \text{ iff } \mathcal{S}_2 \subseteq \mathcal{S}_1 \text{ and } \mathcal{U}_2 \supseteq \mathcal{U}_1 \cup (\mathcal{S}_1 \setminus \mathcal{S}_2).$$

Let $\sqcup$ be the least upper bound (lub) induced by $\sqsubseteq$. It is

$$(\mathcal{S}_1,\ \mathcal{U}_1) \sqcup (\mathcal{S}_2,\ \mathcal{U}_2) = (\mathcal{S}_3,\ \mathcal{U}_3), \text{ with}$$
$$\mathcal{S}_3 \stackrel{def}{=} \mathcal{S}_1 \cap \mathcal{S}_2 \text{ and } \mathcal{U}_3 \stackrel{def}{=} \mathcal{U}_1 \cup \mathcal{U}_2 \cup (\mathcal{S}_1 \Delta\ \mathcal{S}_2),$$

where Δ is the set-theoretic symmetric difference[†].

This definition guarantees that the lub operator adds to $\mathcal{S}$ the variables that are safe in all paths and to $\mathcal{U}$ the variables that are unsafe in at least one path. Given that the operators of union, intersection, and symmetric difference are all both associative and commutative, it follows that the least upper bound operator is also associative and commutative. Moreover, as the union and intersection operators are idempotent and $A \Delta A = \emptyset$, it follows that the least upper bound operator is also idempotent.

We define the abstract semantics of edge labels $(\!|\, l \,|\!) : \wp(\mathbb{V}_q) \times \wp(\mathbb{V}_q) \to \wp(\mathbb{V}_q) \times \wp(\mathbb{V}_q)$, for

[†] Given two sets A and B, $A \Delta B \stackrel{\text{def}}{=} (A \cup B) \setminus (A \cap B)$

each $l \in \mathsf{label}$, as follows:

$$
\begin{aligned}
(\!|\, \texttt{pass} \,|\!)(\mathcal{S},\, \mathcal{U}) &= (\!|\, \vec{x} = \texttt{e} \,|\!)(\mathcal{S},\, \mathcal{U}) = (\mathcal{S},\, \mathcal{U}) \\
(\!|\, \texttt{return}\ \vec{v} \,|\!)(\mathcal{S},\, \mathcal{U}) &= (\mathcal{S} \cup Q(\vec{v}),\, \mathcal{U}) \\
(\!|\, \vec{v}_1 = \mathsf{fun}(\vec{v}_2) \,|\!)(\mathcal{S},\, \mathcal{U}) = (\mathcal{S}_1,\, \mathcal{U}_1) \text{ where } &\begin{cases} \mathcal{S}_1 \stackrel{\text{def}}{=} (\mathcal{S} \setminus Q(\vec{v}_1)) \cup Q(\vec{v}_2) \\ \mathcal{U}_1 \stackrel{\text{def}}{=} (\mathcal{U} \setminus Q(\vec{v}_1)) \end{cases} \\
(\!|\, \texttt{NonZero(b)} \,|\!)(\mathcal{S},\, \mathcal{U}) &= (\!|\, \texttt{Zero(b)} \,|\!)(\mathcal{S},\, \mathcal{U}) = (\mathcal{S} \cup Q(\texttt{b}),\, \mathcal{U}) \\
(\!|\, \texttt{A:}\ \vec{v} \,|\!)(\mathcal{S},\, \mathcal{U}) &= (\mathcal{S} \setminus Q(\vec{v}),\, \mathcal{U} \setminus Q(\vec{v})) \\
(\!|\, \texttt{discard}(\vec{q}) \,|\!)(\mathcal{S},\, \mathcal{U}) &= (\mathcal{S} \cup \{q\},\, \mathcal{U})
\end{aligned}
$$

The first rule deals with a classical instruction and does not change the analysis. In the other rules, the quantum variables which are used are added to $\mathcal{S}$ since all the uses consume variables. The variables $\vec{v}_1$ and $\vec{v}$ are removed from both $\mathcal{S}$ and $\mathcal{U}$ in the third and fifth rule, since these instructions define the variables, making them no more live. Since each statement consumes the variable, variables are never added to $\mathcal{U}$ by the abstract semantics. Nevertheless, this set will be updated by the join operator between paths.

Proposition 4.2. *The abstract semantics $(\!|\, \cdot \,|\!) : \wp(\mathbb{V}_q) \times \wp(\mathbb{V}_q) \to \wp(\mathbb{V}_q) \times \wp(\mathbb{V}_q)$ is monotonic w.r.t. $\sqsubseteq$.*

Proof. We have to prove that if $(\mathcal{S}_1,\, \mathcal{U}_1) \sqsubseteq (\mathcal{S}_2,\, \mathcal{U}_2)$ then $(\!|\, l \,|\!)(\mathcal{S}_1,\, \mathcal{U}_1) \sqsubseteq (\!|\, l \,|\!)(\mathcal{S}_2,\, \mathcal{U}_2)$. Since for all possible l, $(\!|\, l \,|\!)$ changes $(\mathcal{S}_1,\, \mathcal{U}_1)$ and $(\mathcal{S}_2,\, \mathcal{U}_2)$ in the same ways, the proof follows trivially by the definition of the abstract semantics. □

Since the abstract semantics is monotonic and $\langle \wp(\mathbb{V}_q) \times \wp(\mathbb{V}_q), \sqsubseteq \rangle$ is a finite domain (as $\mathbb{V}_q$ is finite for all programs), we are guaranteed that by iteratively applying the equations, we reach a fix-point.

Proposition 4.3. *The abstract semantics $(\!|\, \cdot \,|\!) : \wp(\mathbb{V}_q) \times \wp(\mathbb{V}_q) \to \wp(\mathbb{V}_q) \times \wp(\mathbb{V}_q)$ is distributive, i.e., for all labels l and sets $\mathbf{X} \subseteq \wp(\mathbb{V}_q) \times \wp(\mathbb{V}_q)$, we have*

$$(\!|\, l \,|\!)(\sqcup \mathbf{X}) = \bigsqcup \{ (\!|\, l \,|\!)(\mathcal{S},\, \mathcal{U}) \mid (\mathcal{S},\, \mathcal{U}) \in \mathbf{X} \}.$$

Proof. Thanks to the associativity of the lub operator, we need only to prove that, given $X = (\mathcal{S}_1,\, \mathcal{U}_1)$ and $Y = (\mathcal{S}_2,\, \mathcal{U}_2)$, then $(\mathcal{S}_3,\, \mathcal{U}_3) \stackrel{\text{def}}{=} (\!|\, \cdot \,|\!)(X \sqcup Y) = (\!|\, \cdot \,|\!)(X) \sqcup (\!|\, \cdot \,|\!)(Y) \stackrel{\text{def}}{=} (\mathcal{S}_4,\, \mathcal{U}_4)$. The generic abstract semantics, for any $l \in \mathsf{label}$, on a pair $(\mathcal{S},\, \mathcal{U})$ can be defined as $(\!|\, l \,|\!)(\mathcal{S},\, \mathcal{U}) = ((\mathcal{S} \setminus K) \cup G, (\mathcal{U} \setminus K') \cup G')$ where $K, G, K', G' \in \wp(\mathsf{Vars})$ depend on l. Firstly,

we consider $\mathcal{S}_1$ and $\mathcal{S}_2$, i.e., the safe variable sets:

$$\mathcal{S}_3 = ((\mathcal{S}_1 \cap \mathcal{S}_2) \setminus K) \cup G = ((\mathcal{S}_1 \setminus K) \cup G) \cap ((\mathcal{S}_2 \setminus K) \cup G) = \mathcal{S}_4$$

Now we consider $\mathcal{U}_1$ and $\mathcal{U}_2$, i.e., the unsafe variables are:

$$\begin{aligned} \mathcal{U}_3 &= (((\mathcal{U}_1 \cup \mathcal{U}_2) \setminus K) \cup G) \cup (\mathcal{S}_1 \Delta\ \mathcal{S}_2) \\ &= ((\mathcal{U}_1 \setminus K) \cup G) \cup ((\mathcal{U}_1 \setminus K) \cup G) \cup (\mathcal{S}_1 \Delta\ \mathcal{S}_2) = \mathcal{U}_4 \end{aligned}$$

□

To compute the analysis on the CFG, we need to compute the pair $(\mathcal{S},\ \mathcal{U})$ for each node of the CFG [SWH12]. Similarly to standard liveness, our analysis is backward, i.e., the pair $(\mathcal{S},\ \mathcal{U})$ at node u depends on the pairs $(\mathcal{S}',\ \mathcal{U}')$ of its successors and the label semantics of the edges exiting from u. Given a CFG G, for all node v in G, we define the following system of equations:

$$(\mathcal{S},\ \mathcal{U})[u] \begin{cases} (\varnothing, \varnothing) & \text{if } u = \textbf{end} \\ \bigsqcup \{\ (\!|\, l \,|\!)((\mathcal{S},\ \mathcal{U})[v]) \mid (u, l, v) \in G \} & \text{otherwise} \end{cases} \tag{4.5}$$

As we did for the consuming analysis, this system can be solved by fix-point, also obtaining, in this case, the so-called MFP solution [SWH12]. Due to Proposition 4.3 [Aho+06, Chapter 9][NNH99, Chapter 2], this solution provides the best solution we can compute on CFG, which is the MOP solution computed on the graph nodes, i.e.,

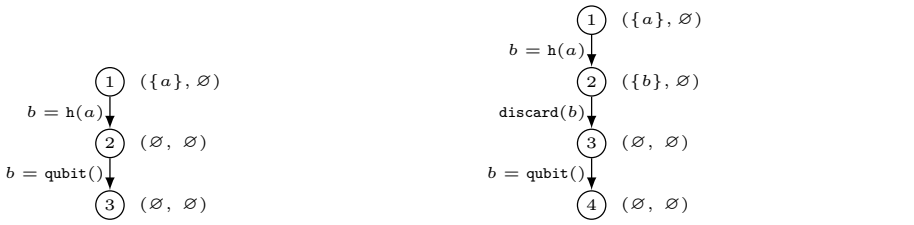
$$(\mathcal{S},\ \mathcal{U})^*[v] = \bigsqcup \left\{ (\!|\, \pi \,|\!)(\mathcal{S}_{\mathbf{e}},\ \mathcal{U}_{\mathbf{e}}) \;\middle|\; \begin{array}{l} \pi = k_1, \ldots, k_n \text{ is a path from } v \text{ to } \textbf{end}, \\ (\!|\, \pi \,|\!) \stackrel{\text{def}}{=} (\!|\, k_1 \,|\!) \circ \cdots \circ (\!|\, k_n \,|\!), \end{array} \right\}$$

Where **end** is the final node of the control flow graph, i.e., the program exit point, and $(\mathcal{S}_{\mathbf{e}},\ \mathcal{U}_{\mathbf{e}}) = \bot$ is the pair $(\mathcal{S},\ \mathcal{U})$ holding the **end** point.

Soundness. By construction, $\mathcal{U} \cap \mathcal{S} = \varnothing$, and if we join $\mathcal{U}$ and $\mathcal{S}$, we obtain the set of live variables. Hence, being the liveness analysis sound, if a variable x is live in a point u, then $x \in \mathcal{U} \cup \mathcal{S}$, with $(\mathcal{S},\ \mathcal{U}) = (\mathcal{S},\ \mathcal{U})[u]$.

Proposition 4.4. *For each program point u, if x is unsafe live in u, then $x \in \mathcal{U}$, with $(\mathcal{S},\ \mathcal{U}) = (\mathcal{S},\ \mathcal{U})[u]$.*

Proof. Consider a node u and a live variable x at u. By definition, x is unsafe live if it is not consumed in at least one path from node u to the end node. Therefore, in at least one



(a) CFG corresponding to Figure 4.4a (b) CFG corresponding to Figure 4.4b

Figure 4.7: In both CFGs, on the right, the pairs $(\mathcal{S}, \mathcal{U})$

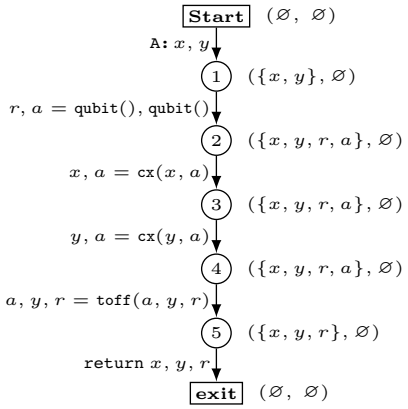


Figure 4.8: CFG corresponding to Figure 4.3a, on the rights the pairs $(\mathcal{S}, \mathcal{U})$

path, the abstract semantics does not add x in $\mathcal{S}$, so when applying the join on paths, x will be added to $\mathcal{U}$. □

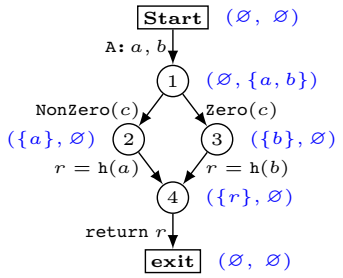
The analysis is incomplete because the MOP solution considers all feasible paths, including those that may never be executed. Nonetheless, this is not an issue since uncomputation is placed only in unsafe paths. If unfeasible paths make the variable unsafe, the uncomputation is also placed in those paths and thus will never be performed.

Examples. First, we consider the code in Figure 4.4a that we represent as a CFG accompanied by the analysis results depicting $(\mathcal{S}, \mathcal{U})$ for each node in Figure 4.7a. In Figure 4.7a, where the `discard` is needed, the variable b is not live. Instead, if we analyse the correct versions of the program (Figure 4.4b and Figure 4.7b), we see that after the definition, the variable b is live, thanks to the discard function.

Consider the code in Figure 4.3a. Figure 4.8 shows the CFG corresponding to the program and the analysis results. After node 4, the variable a is no longer live; thus, in Figure 4.3b, we insert the uncomputation just after edge $(4, a, y, r = \texttt{toff}(a, y, r), 5)$. The uncomputation

```
def fun(a: qubit,b: qubit,c: bool) -¿
    qubit:
  # c is classical
  if c:
     r = h(a)
  else:
     r = h(b)
  return r
```

(a)



(b)

Figure 4.9: The function (a) consumes different variables in different branches. In particular, the `if − branch` consumes only `a`, thus implicitly discarding `b`, while the `else − branch` consumes `b`, thus implicitly discarding `a`. In (b), we show the results of the uncomputation analysis on the CFG of the function (a) and how it detects `a, b` as unsafe at node 1.

analysis is very useful when a variable needs to be discarded only in one branch or when an overwrite occurs in a loop. As an example, consider the program in Figure 4.9a. Since `a` and `b` are used only in one branch, they are unsafe. In fact, when we execute the if-branch, we implicitly discard `b`, and when we execute the else-branch, we implicitly discard `a`. In this case, a simple analysis that detects only the live variable is not enough. Instead, as we see in Figure 4.9b, due to the lub operator, our analysis inserts both a and b in the set $\mathcal{U}[1]$.

4.3 Applying the Analyses to Quantum Programs

It should be clear that the main motivation for the design of the analyses presented above lies in exploiting the information they provide to transform the program automatically without requiring programmer actions. Hence, after formally defining the analyses, it becomes fundamental to define the appropriate pipeline in which they should be performed. This is represented in Figure 4.10 for a given program CFG and consists of four stages:

(1) *Consuming Analysis.* We first perform this analysis to obtain the sets $\mathcal{D}[u]$ for each node u, over-approximating the sets of variables available in u.
(2) *Consuming check.* We use $\mathcal{D}$ to check whether the program is correct, i.e., whether it does not use consumed or undefined variables.
(3) *Uncomputation Analysis.* On correct programs, we perform the uncomputation analysis.
(4) *Discard insertion.* The results of such analysis are then used to decide where it is recommended to insert a discard, avoiding implicit discard operations.

The first and the third steps are precisely the analyses described in the previous section. What we need now to define precisely are the procedures for performing the second and

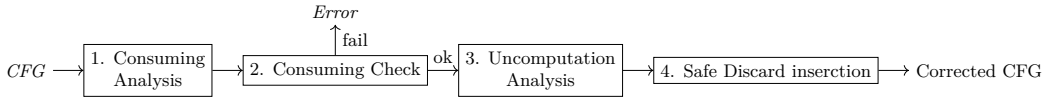


Figure 4.10: Analysis pipeline

fourth steps of the pipeline.

Consuming Check. After performing the Consuming Analysis, we obtain the sets $\mathscr{D}[u]$ for each program point. First, we check if every used variable is defined and not consumed. Moreover, we also have to check that a variable is not passed more than once as an argument (e.g., the statement $\mathsf{fun}(q, q)$ is discarded since it introduces implicit copies).

Algorithm 1 Check for Variable Usage

Require: *edges*: control flow graph edges, $\mathscr{D}$
1: **for** u, v **in** *edges* **do**
2: $label \leftarrow$ GETLABEL(u, v)
3: **if** *label* is a function call **then**
4: **if** $\neg$ CHECKLEGITUSE(*label*) **then**
5: **return** 'Error: implicit copy'
6: **end if**
7: **end if**
8: $usedVars \leftarrow$ GETUSEDVARS(*label*)
9: **if** $(usedVars \nsubseteq \mathscr{D}[u])$ **then**
10: **return** 'Error, not defined variable used.'
11: **end if**
12: **end for**

As shown in Algorithm 1, to check this property, for all edges (u, l, v), i.e., for all labels l in the CFG, we get the set of all used variable ($usedVars$), and if this set is not included in $\mathscr{D}[u]$ this means that there are some variables that are used without being defined or after being consumed, so we return an error. In Algorithm 1, we simplified the returned message in case of an error. In the implemented version, we return a detailed error specifying which variables generate errors and at which point in the program. For instance, the example in Figure 4.6 shows that the check fails when considering the edge $(2, b = \mathtt{h}(a), 3)$ since a is not in $\mathscr{D}[2]$.

Discard Insertion. The final step involves inserting the `discard` function. We must discard all variables that were not consumed or returned in at least one path, i.e., defined and never used (not live) or used but not in all paths. Hence, in the first case, we can identify all non-live variables and insert the uncomputation just after their definition. In

the second case, the variables, at some program point, are in the set $\mathcal{U}$ determined by the Uncomputation analysis, and we have to insert the discard only in those paths that do not consume it. This means that, in the second case, we must check if (and where) a variable is in $\mathcal{U}$ to understand where we have to insert a discard. We show the algorithm in detail in Algorithm 2.

Algorithm 2 Insert discard

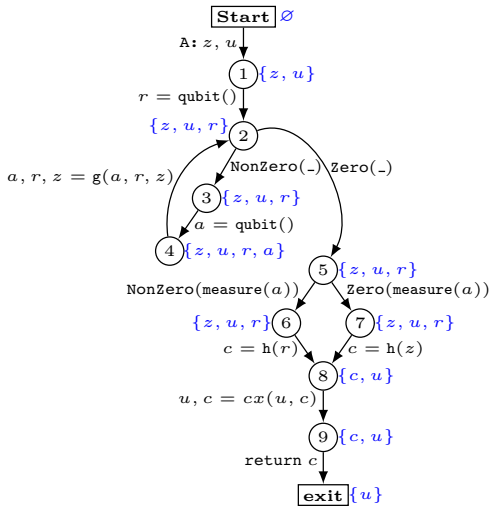
Require: $edges, nodes, (\mathcal{S},\ \mathcal{U}), \mathbb{V}_q$
1: **for** var **in** $\mathbb{V}_q$ **do**
2: **for** u, v **in** GETALLDEFINITION(cfg, var) **do**
3: **if** $var \notin (\mathcal{S}[v] \cup \mathcal{U}[v])$ **then**
4: ADDDISCARD$((u, v), var)$
5: **end if**
6: **end for**
7: **end for**
8: **for** var **in** $\bigcup_u \mathcal{U}[u]$ **do**
9: **for** $node$ **in** $nodes$ **do**
10: **if** $var \in \mathcal{U}[node]$ **then**
11: **for** v **in** SUCCESSOR$(node)$ **do**
12: **if** $var \notin (\mathcal{S}[v] \cup \mathcal{U}[v])$ **then**
13: ADDDISCARD$((node, v), var)$
14: **end if**
15: **end for**
16: **end if**
17: **end for**
18: **end for**

The procedure receives as input the set of arcs and nodes from the CFG, the list pairs $(\mathcal{S},\ \mathcal{U})$ from the previous analysis, and the program quantum variables set $\mathbb{V}_q$. First, for each quantum variable $var \in \mathbb{V}_q$, we check if the variable is defined and not used, verifying if it is live after its definition. If not, we must insert the discard after the definition. Then, we consider all variables that, for some nodes u, are in the sets $\mathcal{U}[u]$. If var is in some $\mathcal{U}[u]$, there is some node in which, in some of the paths that start in that node, the variable is safe live, and others in which it is not live. So, for each node u of the CFG, if x is in the set $\mathcal{U}[u]$, we check the successors of u. For each successor v of u, if x is not in the set $\mathcal{S}[v]$ and is not in the set $\mathcal{U}[v]$, the node v is the head of the 'unsafe' path, so a discard operation is added between nodes u and v.

In Algorithm 2, we apply our algorithm to the simple examples introduced so far. Let us consider the example in Figure 4.7a. Here the assignment in edge $(1, b = \mathtt{h}(a), 2)$ defines a non-live variable (b). So we insert the discard at the end of that edge (just like we did in Figure 4.4b and in Figure 4.7b). Similarly, for the example in Figure 4.8, when we apply

```
1 def f(z:qubit,u:qubit,n:int):
2     r = qubit()
3     for _ in range(n):
4         a = qubit()
5         a,r,z = g(a,r,z)
6
7     if measure(a):
8         c = h(r)
9     else:
10        c = h(z)
11
12    u,c = cx(u,c)
13
14    return c
```

(a)



(b)

Figure 4.11: The example function

the discard insertion, the only assignment that defines a non-live variable is the one in edge $(4, a, y, r = \texttt{toff}(a, y, r), 5)$, so we insert the discard at the end of that edge (line 6 in Figure 4.3a, just like the function in Figure 4.3b shows). Now consider the function analysed in Figure 4.9b. In this case, there are no non-living variables, but two variables are in $\mathcal{U}$. Applying the discard insertion algorithm, we select the edge $(1, \texttt{NonZero}(c), 2)$ to insert the discard of b and $(1, \texttt{Zero}(c), 3)$ to insert the discard of a.

Evaluation. We have implemented a prototype of our procedure in Python 3[‡]. We tested both the consuming check and the insertion of the discard operation, paying more attention to the discard insertion. In particular, the discard insertion has been tested on ten simple programs that involve the redefinition of unconsumed variables within loops, redefinition in different computational branches, and the presence of unused variables. Moreover, all the examples in this chapter have been tested.

4.4 A Complete Example

In this section, we apply the entire pipeline to a more complex program, detailing the system evolution that leads to the MFP solution. Consider the function f in Figure 4.11a and the corresponding CFG in Figure 4.11b. We show the system of equations derived from the CFG in Figure 4.12 and the system's computation in Figure 4.12b. In Figure 4.11b, we also

‡ The code available at https://github.com/NicolaAssolini98/StaticAnalysisQuantumPrograms.

$\mathcal{D}[\textbf{Start}] = \varnothing$
$\mathcal{D}[1] = \mathcal{D}[\textbf{Start}] \cup \{z, u\}$
$\mathcal{D}[2] = (\mathcal{D}[1] \cup \{r\}) \cap (\mathcal{D}[4] \setminus \{a, r, z\} \cup \{a, r, z\})$
$\mathcal{D}[3] = \mathcal{D}[2]$
$\mathcal{D}[4] = \mathcal{D}[3] \cup \{a\}$
$\mathcal{D}[5] = \mathcal{D}[2]$
$\mathcal{D}[6] = \mathcal{D}[5] \setminus \{a\}$
$\mathcal{D}[7] = \mathcal{D}[5] \setminus \{a\}$
$\mathcal{D}[8] = (\mathcal{D}[6] \setminus \{r\} \cup \{c\}) \cap (\mathcal{D}[6] \setminus \{z\} \cup \{c\})$
$\mathcal{D}[9] = (\mathcal{D}[8] \setminus \{u, c\} \cup \{u, c\})$
$\mathcal{D}[\textbf{End}] = \mathcal{D}[9] \setminus \{c\}$

(a)

	0	1	F.P.
Start	$\varnothing$	$\varnothing$	$\varnothing$
1	$\mathbb{V}_q$	$\{z, u\}$	$\{z, u\}$
2	$\mathbb{V}_q$	$\{z, u, r\}$	$\{z, u, r\}$
3	$\mathbb{V}_q$	$\{z, u, r\}$	$\{z, u, r\}$
4	$\mathbb{V}_q$	$\{z, u, r, a\}$	$\{z, u, r, a\}$
5	$\mathbb{V}_q$	$\{z, u, r\}$	$\{z, u, r\}$
6	$\mathbb{V}_q$	$\{z, u, r\}$	$\{z, u, r\}$
7	$\mathbb{V}_q$	$\{z, u, r\}$	$\{z, u, r\}$
8	$\mathbb{V}_q$	$\{u, c\}$	$\{u, c\}$
9	$\mathbb{V}_q$	$\{u, c\}$	$\{u, c\}$
End	$\mathbb{V}_q$	$\{u\}$	$\{u\}$

(b)

Figure 4.12: We show in (a) the system of equations derived from Figure 4.11b and in (b) the MFP solution of the system

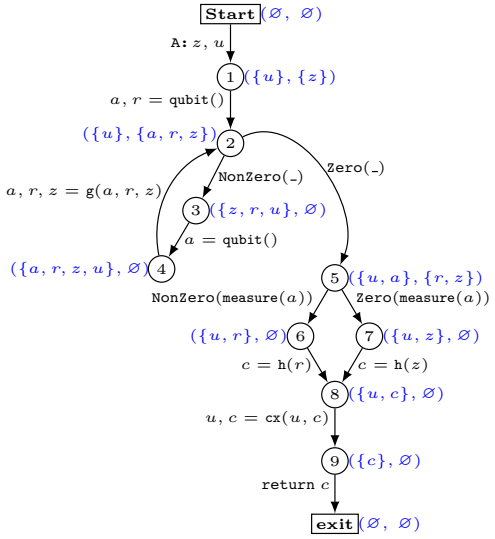
indicate the sets $\mathcal{D}$ for each program point computed by Consuming Analysis. Then, applying the algorithm in Algorithm 1, we raise an error in edges $(5, \texttt{NonZero}(\texttt{measure}(a)), 6)$ and $(5, \texttt{Zero}(\texttt{measure}(a)), 7)$ since we use a, which is not defined in all paths. Figure 4.13a and Figure 4.13b show the code and the CFG of the function after correcting the errors highlighted by the previous step. Now, this function passes the correctness check, and we can analyse it to see if it needs some discard functions. We show the system of equations derived from the CFG in Figure 4.14a and the system's computation in Figure 4.14b. In Figure 4.13b, we show on each node the sets $(\mathcal{S}, \mathcal{U})$ corresponding to the MFP solution computed in Figure 4.14b. We now apply the algorithm in Algorithm 2. The variable u is not live after its last definition in the edge $(8, u, c = \texttt{cx}(u, c), 9)$, so we insert the discard there. The variable a is in $\mathcal{U}[2]$, being in $\mathcal{S}[5]$ and not being live in 3, we insert $\texttt{discard}(a)$ in $(2, \texttt{NonZero}(\_), 3)$. The variable r is Unsafe in multiple nodes, in particular:

- $r \in \mathcal{U}[2]$ but is live in both 3 and 5, so we do not need to do anything;
- $r \in \mathcal{U}[5]$ and being in $\mathcal{S}[6]$ and not being live in 7, we insert $\texttt{discard}(\text{r})$ in the edge $(5, \texttt{Zero}(\texttt{measure}(a)), 7)$.

Similarly, z is in $\mathcal{U}[2]$ and live in the successors, so it does not need to be discarded, whereas it is in $\mathcal{U}[5]$ and in $\mathcal{S}[7]$ and not live in 6, we insert $\texttt{discard}(z)$ in $(5, \texttt{NonZero}(\texttt{measure}(a)), 6)$. The procedure employed in this example successfully resolved the implicit discarding, which would have resulted in the type system rejecting the program. Finally, we present the output programs with the discard insertions in Figure 4.15. We note that by using a type-based approach, the type checker rejects the program in Figure 4.13a unless the programmer enters all the discard functions shown in Figure 4.15.

```
1 def r˙f(z:qubit,u:qubit,n:int):
2     a, r = qubit()
3     for ˙ in range(n):
4         a = qubit()
5         a,r,z = g(a,r,z)
6
7     if measure(a):
8         c = h(r)
9     else:
10        c = h(z)
11
12    u,c = cx(u,c)
13
14    return c
```

(a)



(b)

Figure 4.13: The correct version of function `f` in Figure 4.11

$(\mathcal{S}, \mathcal{U})[\textbf{End}] = (\varnothing, \varnothing)$
$(\mathcal{S}, \mathcal{U})[9] = (\mathcal{S}[\textbf{End}] \cup \{c\}, \mathcal{U}[\textbf{End}])$
$(\mathcal{S}, \mathcal{U})[8] = (\mathcal{S}[9] \setminus \{u,c\} \cup \{u,c\}, \mathcal{U}[9] \setminus \{u,c\})$
$(\mathcal{S}, \mathcal{U})[7] = (\mathcal{S}[8] \setminus \{c\} \cup \{z\}, \mathcal{U}[8] \setminus \{c\})$
$(\mathcal{S}, \mathcal{U})[6] = (\mathcal{S}[8] \setminus \{c\} \cup \{r\}, \mathcal{U}[8] \setminus \{c\})$
$(\mathcal{S}, \mathcal{U})[5] = (\mathcal{S}[6] \cup \{a\}, \mathcal{U}[6]) \sqcup (\mathcal{S}[7] \cup \{a\}, \mathcal{U}[7])$
$(\mathcal{S}, \mathcal{U})[4] = (\mathcal{S}[2] \setminus \{a,r,z\} \cup \{a,r,z\}, \mathcal{U}[2] \setminus \{a,r,z\})$
$(\mathcal{S}, \mathcal{U})[3] = (\mathcal{S}[4] \setminus \{a\}, \mathcal{U}[4] \setminus \{a\})$
$(\mathcal{S}, \mathcal{U})[2] = (\mathcal{S}, \mathcal{U})[3] \sqcup (\mathcal{S}, \mathcal{U})[5]$
$(\mathcal{S}, \mathcal{U})[1] = (\mathcal{S}[2] \setminus \{a,r\}, \mathcal{U}[2] \setminus \{a,r\})$
$(\mathcal{S}, \mathcal{U})[\textbf{Start}] = (\mathcal{S}[1] \setminus \{z,u\}, \mathcal{U}[1] \setminus \{z,u\})$.

(a) System of equations derived from the CFG in Figure 4.13b

	0	1	2	F.P.
End	$(\varnothing, \varnothing)$	$(\varnothing, \varnothing)$	$(\varnothing, \varnothing)$	
9	$\bot$	$(\{c\}, \varnothing)$	$(\{c\}, \varnothing)$	
8	$\bot$	$(\{u,c\}, \varnothing)$	$(\{u,c\}, \varnothing)$	
7	$\bot$	$(\{u,z\}, \varnothing)$	$(\{u,z\}, \varnothing)$	
6	$\bot$	$(\{u,r\}, \varnothing)$	$(\{u,r\}, \varnothing)$	
5	$\bot$	$(\{u,a\}, \{r,z\})$	$(\{u,a\}, \{r,z\})$	
4	$\bot$	$\bot$	$(\{a,r,z,u\}, \varnothing)$	
3	$\bot$	$\bot$	$(\{z,r,u\}, \varnothing)$	
2	$\bot$	$(\{u,a\}, \{r,z\})$	$(\{u\}, \{a,r,z\})$	
1	$\bot$	$(\{u\}, \{z\})$	$(\{u\}, \{z\})$	
Start	$\bot$	$(\varnothing, \varnothing)$	$(\varnothing, \varnothing)$	

(b) MFP solution of the system

Figure 4.14

```
def rr`f(z:qubit,u:qubit,n:int):
   a, r = qubit()
   for ` in range(n):
      discard(a)
      a = qubit()
      a,r,z = g(a,r,z)
   if measure(a):
      discard(z)
      c = h(r)
   else:
      discard(r)
      c = h(z)
   u,c = cx(u,c)
   discard(u)

   return c
```

Figure 4.15: The function in Figure 4.13 after the discard insertion

4.5 Related Works

Here we recall some work related that are not covered in the survey in Chapter 3.

Most of the literature has focused on synthesising uncomputation and optimising the efficiency of the resulting procedure, rather than detecting which variables or qubits actually need to be uncomputed (i.e., discarded). Examples include [PRS17; AMK15; ARS17; Par+21; PBV24; VG25; Rei+25; Tiw+25].

Square [Din+20] and `staq`[AG20] use uncomputation to reduce the number of ancilla qubits. Quipper[Gre+13a], instead, automatically converts classical Haskell programs into quantum circuits and introduces uncomputation during this process. Similarly, Qunity [Voi+23] offers automatic uncomputation when compiling high-level controlled operations in circuits. Qrisp [Sei+24; Sei+23] allows automatic uncomputation through an `@auto_uncompute` decorator, and manual uncomputation with an explicit `uncompute` function. Qrisp operates at a slightly higher level than circuit languages, offering an abstraction via a `QuantumVariable` class (and subclasses), which still represents sets of qubits and works positionally (e.g., `q = QuantumVariable(1); x(q)`). Finally, the recent language Quff[Wri+24], published at the same time as our work[ADM24b], addresses a similar goal. Quff automatically triggers uncomputation whenever a quantum variable goes out of scope at runtime. Unlike our approach, however, it does not provide a formal definition of the method.

4.6 Discussion

In this chapter, we have introduced an analysis of *uncomputation* for high-level quantum programming languages, aimed at reducing compilation errors and relieving programmers

```
1  a = qubit()
2  if ·:
3     c = f(a)
4  else:
5     c = qubit()
6  return c
```

Figure 4.16

from low-level resource management. As shown in Figure 4.10, the approach combines two sequential analyses that overcome key limitations of type-system-based methods: a forward analysis ensuring the single use of quantum variables, followed by a backward analysis that identifies where variables should be discarded (*uncomputed*).

Our analysis introduces the same level of approximation as a type checker. Both the analysis and the type system consider all possible paths, even those that are not executable. One crucial distinction between the two approaches is that the type system will generate an error, forcing the programmer to modify an infeasible path. In contrast, our method will automatically modify only the impossible path, ensuring that the program semantics remain unaffected. Moreover, our static analysis is more informative than the type system, as the example in Figure 4.16 highlights. A type checker based on linear types would return an error at line 1, indicating that `a` is not consumed but giving no information on the point where the discard must be inserted, namely the `else` − `branch`. Instead, our approach would identify that the problem lies in the `else`-branch and would pass the information to the next step of the pipeline without rejecting the program. Consequently, even when using our approach only as a linearity checker, it would provide more informative results than the type system. This is particularly beneficial when the definition and the path that does not use the variable are far apart in the code, and the control flow is complicated. Moreover, Replacing linear typing with static analysis increases flexibility—it is language-independent and can integrate with other analyses to improve precision, e.g., through classical interval or constant-propagation analyses, or quantum-specific ones such as entanglement analysis [Per07; Per08b; Hon15; Jav+14]. The method applies to languages with or without linear type systems, such as Quipper [Gre+13a], and can be adapted to those using annotations like Silq [Bic+20]. By leveraging modified liveness information, it supports non-consuming commands while still detecting variables to uncompute, ensuring broad applicability across quantum programming languages.

Chapter 5

A Static Analysis of Entanglement

Quantum computation is characterised by two fundamental principles: superposition and entanglement. Unlike classical systems, where a state exists in a single, definite configuration, a quantum state can simultaneously exist in a superposition of multiple classical states. Entanglement occurs when some particles become so strongly correlated through a computation that they form an 'inseparable' state, i.e., a state where they are no longer identifiable in their individual states. For program variables, this means that every time we act on a variable q, we also alter the state of the other variables entangled with q. Entanglement is crucial for many applications, such as quantum communication protocols (e.g., quantum teleportation [KLM06, Chapter 5]), but can be problematic in quantum programming, particularly when combined with the principle of implicit measurement [NC10, Chapter 4]. Therefore, analysing and tracking entanglement is essential for effectively reasoning about quantum programs. A particularly important task is identifying sets of entangled variables, namely sets of variables, such that whenever we operate on one variable, we may alter other variables in the same set. Crucial for this task is to identify which variables are in a quantum state and which are not.

5.1 A Minimal Quantum Language

To define our analysis, we refer to a minimal quantum language characterised by quantum statements on quantum variables with a control flow based on measurement. Similarly to what we have done in Section 4.1, we introduce our minimal quantum language as a CFG language. This means that the branching in the CFG is guided by the probabilistic result of

a quantum measurement. Given a finite set of quantum variables $\mathbb{V}_q$ and $q, p \in \mathbb{V}_q$ we define the language label as follows:

$$\begin{aligned} \mathtt{c} &::= \mathsf{label} \mid \mathtt{c}; \mathtt{c} \mid \mathtt{c} \oplus \mathtt{c} \mid \mathtt{c}^* \\ \mathsf{label} &::= \mathtt{M}_1(q) \mid \mathtt{M}_0(q) \mid \mathtt{skip} \mid \mathtt{h}(q) \mid \mathtt{t}(q) \mid \mathtt{cx}(p,q), \end{aligned} \tag{5.1}$$

where $\mathtt{M}_0(q)$ ($\mathtt{M}_1(q)$) indicates that the measurement of q returns 0 (1); the statements, $\mathtt{h}$, $\mathtt{t}$, and $\mathtt{cx}$ indicate, respectively, the Hadamard gate, the T gate and the control-not gate [NC10, Chapter 4]. Considering only these three quantum operations is not a limitation because they allow us to cover all possible quantum computations [Boy+00]. Moreover, without loss of generality, we can assume that every variable $q_i \in \mathbb{V}_q$ corresponds to a 1-qubit register initialised to the state $|0\rangle$. As we show in Equation 4.2, our language is equivalent to the quantum while language used in [Yin12; Yin16; PYW22], and to the one that is used to define the other two entanglement analyses based on abstract semantics [Hon15; Per08b].

Collecting semantics

Now, we can define the concrete collecting semantics of our CFG language. Let $Q = \{q_i\}_n$ be the set of variables, we call $\mathscr{H}_{\mathbb{V}_q} = \bigotimes_i^n \mathscr{H}_{q_i}$ the n-qubit Hilbert space, i.e., a space of dimension 2^n. Let $V_{\mathbb{V}_q}$ be the set of all vectors $|\psi\rangle \in \mathscr{H}_{\mathbb{V}_q}$, we define the collecting semantics as a function $[\![\cdot]\!] : \wp(V_{\mathbb{V}_q}) \to \wp(V_{\mathbb{V}_q})$. First, given a set $v \in \wp(V_Q)$, we define the collecting semantics for each instruction of the language label as follows:

$$\begin{aligned} [\![\,\mathtt{skip}\,]\!]v &= v \\ [\![\,\mathtt{h}(q)\,]\!]v &= \{\ H_q\,|\psi\rangle \mid |\psi\rangle \in v\,\} \\ [\![\,\mathtt{t}(q)\,]\!]v &= \{\ T_q\,|\psi\rangle \mid |\psi\rangle \in v\,\} \\ [\![\,\mathtt{cx}(p,q)\,]\!]v &= \{\ CX_{p,q}\,|\psi\rangle \mid |\psi\rangle \in v\,\} \\ [\![\,\mathtt{M}_1(q)\,]\!]v &= \left\{ \frac{M_{1q}\,|\psi\rangle}{\|M_{1q}\,|\psi\rangle\,\|} \;\middle|\; \|M_{1q}\,|\psi\rangle\,\|^2 > 0, |\psi\rangle \in v \right\} \\ [\![\,\mathtt{M}_0(q)\,]\!]v &= \left\{ \frac{M_{0q}\,|\psi\rangle}{\|M_{0q}\,|\psi\rangle\,\|} \;\middle|\; \|M_{0q}\,|\psi\rangle\,\|^2 > 0, |\psi\rangle \in v \right\} \end{aligned}$$

where H_q and T_q are the unitary operators in $\mathscr{H}_{\mathbb{V}_q}$ that correspond to the gate Hadamard and T applied to q, $CX_{p,q}$ is the unitary that corresponds to control-not on p and q (where p is the controller and q the target) and the identity on the other variables while $\mathtt{M}_1(q)$ and $\mathtt{M}_0(q)$ corresponds to measurement 1 and 0 on q. For instance $[\![\,\mathtt{M}_1(q)\,]\!]\{|1\rangle_q\} = \{|1\rangle_q\}$ while

$[\![\mathtt{M}_0(q)]\!]\{|1\rangle_q\} = \varnothing$. Finally, we can define the collecting semantics for the whole language:

$$\begin{aligned} [\![\mathtt{c}_1; \mathtt{c}_2]\!] v &= [\![\mathtt{c}_2]\!]([\![\mathtt{c}_1]\!] v) \\ [\![\mathtt{c}_1 \oplus \mathtt{c}_2]\!] v &= [\![\mathtt{c}_1]\!] v \cup [\![\mathtt{c}_2]\!] v \\ [\![\mathtt{c}^*]\!] v &= \bigcup_n [\![\mathtt{c}^n]\!] v. \end{aligned} \tag{5.2}$$

With this collecting semantics, we only want to represent the set of all states that a program can return as a result. For this reason, when we consider the measurement, we follow a conservative approach by collecting all possible results, ignoring the probability with which these results occur.

5.2 Characterising Entanglement

In this section, we define the properties of *separability* and of *direct inseparability*, and the abstract domain proposed to represent them. Here, the idea is to refine this domain to make it suitable for performing a static analysis for soundly detecting entanglement. To define an abstract domain which is able to capture the entanglement property of quantum variables, we introduce a characterisation of this property by means of an equivalence relation. For bipartite systems (e.g., two qubits), entanglement and separability are dual concepts. In fact, a composite quantum state $|\psi\rangle_{q_1,q_2} \in \mathscr{H}_{q_1} \otimes \mathscr{H}_{q_2}$ is separable if and only if it can be written as a tensor product $|\psi\rangle_{q_1,q_2} = |\phi_{q_1}\rangle \otimes |\phi_{q_2}\rangle$ for some states $|\phi_{q_1}\rangle \in \mathscr{H}_{q_1}$ and $|\phi_{q_2}\rangle \in \mathscr{H}_{q_2}$. A state $|\psi\rangle_{q_1,q_2}$ is entangled if and only if it is *not* separable. However, the scenario becomes more complex when considering systems consisting of three or more subsystems. Entanglement in such systems corresponds to inseparability across the entire system, but various degrees of entanglement can occur within subsystems.

Some metrics have been introduced to analyse the entanglement of these systems, such as *entanglement monotones* and *entanglement measures* [Vid00], which quantify entanglement between subsystems. In [Di +18], the entanglement of two subsystems S_1, S_2 is measured in terms of an entanglement monotone function $E_{|\psi\rangle}(S_1, S_2)$, such that $E_{|\psi\rangle}(S_1, S_2) = 0$ if and only if S_1 and S_1 taken in isolation are not entangled in the global system state $|\psi\rangle$. For instance, the 3-qubits state $|\psi\rangle_{q_1,q_2,q_3} = 1/2(|000\rangle + |001\rangle + |011\rangle + |111\rangle)_{q_1,q_2,q_3}$ is *fully inseparable* since it cannot be decomposed via the tensor product ($|\psi\rangle = |\phi_1\rangle \otimes |\phi_2\rangle$ for any $|\phi_1\rangle$ and $|\phi_2\rangle$). In fact, the entanglement monotone $E_{|\psi\rangle}(q_i, \{q_j, q_k\})$ always differs from zero for all $i, j, k \in \{1, 2, 3\}$. However, if we measure the entanglement between pairs of qubits, we have $E_{|\psi\rangle}(q_1, q_2) > 0$, $E_{|\psi\rangle}(q_2, q_3) > 0$ but $E_{|\psi\rangle}(q_1, q_3) = 0$, that is the qubits q_1 and q_3 taken in isolation are a separable subsystem. This occurs because tracing out

q_2 yields, with equal probability, either $^1/\sqrt{2}(|00\rangle + |01\rangle)_{q_1,q_3}$ or $^1/\sqrt{2}(|01\rangle + |11\rangle)_{q_1,q_3}$, both corresponding to separable states of q_1 and q_3. This example shows that the entanglement is not transitive, i.e., the fact that q_1 is entangled with q_2 and q_2 with q_3 does not imply that q_1 is entangled with q_3.

In our analysis, we are interested in understanding when a set of variables is fully inseparable or whether the variables are separable in some way. When two variables are separable (and thus not entangled), we know we can measure one without altering the other. Thus, in our analysis, we speak about *separability* and we consider its dual notion *inseparability* instead of entanglement. Let us formally define the separability of two variables in a multi-variable state.

Definition 5.1 (Separability)**.** *Let $\mathbb{V}_q$ be the set of variables in a state $|\psi\rangle_{\mathbb{V}_q}$. Two variables $q_1, q_2 \in \mathbb{V}_q$ are* separable *if the state $|\psi\rangle_{\mathbb{V}_q}$ can be written as $|\psi\rangle_{\mathbb{V}_q} = |\phi_1\rangle_{Q_1} \otimes |\phi_2\rangle_{Q_2}$, where $Q_1, Q_2 \subset \mathbb{V}_q$, $q_1 \in Q_1$ and $q_2 \in Q_2$. Otherwise, we say that q_1, q_2 are* inseparable*. Given a set $v \in \wp(V_{\mathbb{V}_q})$, two variables q_1, q_2 are inseparable in v if they are inseparable in* at least one state $|\psi\rangle_{\mathbb{V}_q} \in v$.

There exists a particular relation between inseparable variables. For instance, let us consider the state

$$|\psi\rangle_{a,b,c} = (|000\rangle + |110\rangle + |001\rangle - |111\rangle)_{a,b,c},$$

where a, b, c are inseparable. On an intuitive level, it is clear that the three variables are not related in the same way. In fact, a and b are more closely related to each other than either a with c or b with c: if we measure a we obtain one of the two states: $(|00\rangle + |01\rangle)_{b,c}$ or $(|10\rangle - |11\rangle)_{b,c}$, where b has collapsed to a base state (0 or 1) in both states while c is still in superposition. Instead, if we measure c we obtain: $(|00\rangle + |11\rangle)_{a,b}$ or $(|00\rangle - |11\rangle)_{a,b}$ where a and b are in a entangled and superpose state. When two variables are related as a and b in this example, we introduce the notion of *directly inseparability.*

Definition 5.2 (Direct Inseparability)**.** *Given $a, b \in \mathbb{V}_q$ in a state $|\psi\rangle_{\mathbb{V}_q}$, we say that a and b are* directly inseparable *(d-inseparable) if, upon measuring one of them, the other collapses to a basis state in all possible measurement outcomes. Two variables $q_1, q_2 \in \mathbb{V}_q$ are d-inseparable in $v \in \wp(V_{\mathbb{V}_q})$ if they are d-inseparable in* all states $|\psi\rangle_{\mathbb{V}_q} \in v$.

Being d-inseparable is a useful property when reasoning about entanglement. In fact, if we apply a controlled not (CX) between two d-inseparable variables, we will always 'disentangle' the target variable. For instance, if we apply $CX_{a,b}$, where a is the controller and b is the

target, the state $|\psi\rangle_{a,b,c}$ defined above, we obtain:

$$\begin{aligned} CX_{a,b}(|\psi\rangle_{a,b,c}) &= (|000\rangle + |100\rangle + (|001\rangle - |101\rangle)_{a,b,c} = \\ &= ((|0\rangle + |1\rangle)\,|0\rangle + (|0\rangle - |1\rangle)\,|1\rangle)_{a,c} \otimes |0\rangle_b\,. \end{aligned} \tag{5.3}$$

In other words, we have separated b from the other variables. Instead if we apply $CX_{a,c}$ we obtain:

$$CX_{a,c}(|\psi\rangle_{a,b,c}) = (|000\rangle + |111\rangle + |001\rangle - |110\rangle)_{a,b,c}, \tag{5.4}$$

and we do not 'disentangle' a since c and a are not d-inseparable.

We now show that the properties of separability and direct inseparability are transitive.

Proposition 5.1. *Given a set of variables $\mathbb{V}_q$ in a state $|\psi\rangle_{\mathbb{V}_q}$ and three variables q_1, q_2 and q_3 in $\mathbb{V}_q$, if q_1 and q_2 are non-separable and q_2 and q_3 are non-separable, then q_1 and q_3 are non-separable.*

Proof. Let $|\psi\rangle_{\mathbb{V}_q} = |\phi_1\rangle_{Q_1} \otimes |\phi_2\rangle_{Q_2}$. Without loss of generality, we can assume that $q_1 \in Q_1$. Since q_1 and q_2 are non-separable q_2 must be in Q_1. But, by hypothesis, also q_2 and q_3 are non-separable, so q_3 must also be in Q_1. This means that q_1 and q_3 must be in the same set (in this case, Q_1), and so they are non-separable. □

Proposition 5.2. *If q_1 and q_2 are d-inseparable and q_2 and q_3 are d-inseparable, then q_1 and q_3 are d-inseparable.*

Proof. If q_1 and q_2 are d-inseparable, then if we measure q_1, then q_2 collapses to a base state, but since q_2 and q_3 are d-inseparable, then also q_3 will collapse. Thus, q_1 and q_3 are d-inseparable. □

Since the non-separability and d-inseparability are transitive, trivially symmetric, and we can easily assume that a variable is non-separable and d-inseparable from itself, i.e., these two properties are reflexive. Thus, inseparability and d-inseparability are equivalence relations.

5.3 An Abstract Domain for Entanglement

The generation of entanglement depends on the values of the variables, which, therefore, must be taken into account when defining the elements of our abstract domain. Thus, we define an abstract state as consisting of two parts: the first is based on sets of variables representing inseparability and d-inseparability. In contrast, the second consists of a function that associates each variable with a specific label indicating the variable's state.

The Inseparability and D-Inseparability Domain

Inseparability and d-inseparability are both equivalence relations; thus, given a set of variables $\mathbb{V}_q$, we can represent both properties on $\mathbb{V}_q$ by partitions of $\mathbb{V}_q$. For instance, let us consider the state

$$|\psi\rangle_{a,b,c,d} = ((|00\rangle + |11\rangle)\,|0\rangle + (|00\rangle - |11\rangle)\,|1\rangle)_{a,b,c} \otimes |1\rangle_d\,.$$

We can build the partition $(\{a,b,c\}\{d\})$ that represents the inseparable variables in $|\psi\rangle_{a,b,c,d}$ and another partition $(\{a,b\},\{c\},\{d\})$ that identifies the d-inseparable variables. Since being d-inseparable implies being inseparable, the d-inseparable partition is always included in the inseparable one. We encode this information in our abstract states by representing them as a list of numbered sets (e.g., $[(\{a,b\},0),(\{c\},0),(\{d\},1)]$), where the smaller partition $(\{a,b\},\{c\},\{d\})$ represents which variables are d-inseparable, while by merging sets with the same number, we obtain the partition representing inseparability $(\{a,b,c\}\{d\})$.

Definition 5.3. *Given a set of quantum variables $\mathbb{V}_q$, we define the abstract state $\mathscr{E}^{\mathbb{V}_q}$ as the set of tuples*

$$\mathscr{E}^{\mathbb{V}_q} = \{\ (e,k)\ |\ e \in \wp(\mathbb{V}_q)\ \textit{and}\ k \in \mathbb{N}\}\,,$$

where $\forall\,(e,k),(e',k') \in \mathscr{E}^{\mathbb{V}_q}, e \cap e' = \varnothing$ and $\bigcup_{(e,k)\in\mathscr{E}^{\mathbb{V}_q}} e = \mathbb{V}_q$. In other words, the sets e form a partition of $\mathbb{V}_q$.

We call $\mathbb{E}^{\mathbb{V}_q} \subset \wp(\wp(\mathbb{V}_q) \times \mathbb{N})$ the abstract domain of all possible $\mathscr{E}^{\mathbb{V}_q}$.

To better refer to the abstract state, we introduce the following notation. Given an abstract state $\mathscr{E}^{\mathbb{V}_q}$, we write E_k, using a capital letter, to refer to the set $E_k = \bigcup_{(e,k)\in\mathscr{E}^{\mathbb{V}_q}} e$, i.e., the union of all e with the same index k. For instance, if

$$\mathscr{E}^{\{a,b,c,d,e\}} = [(\{a,b\},0),(\{c\},0),(\{d,e\},1)],$$

$E_0 = \{a,b,c\}$ while $E_1 = \{d,e\}$. Using this notation, we introduce a partial order in $\mathbb{E}^{\mathbb{V}_q}$.

Definition 5.4 ($\mathbb{E}^{\mathbb{V}_q}, \leqslant_{\mathbb{E}}$)**.** *Given $\mathscr{E}_1^{\mathbb{V}_q}, \mathscr{E}_2^{\mathbb{V}_q} \in \mathbb{E}^{\mathbb{V}_q}$. $\mathscr{E}_1^{\mathbb{V}_q} \leqslant_{\mathbb{E}} \mathscr{E}_2^{\mathbb{V}_q}$ iff $\forall (e_2,k) \in \mathscr{E}_2^{\mathbb{V}_q}$, $\exists\,(e_1,k') \in \mathscr{E}_1^{\mathbb{V}_q}$ such that $e_2 \subseteq e_1$ and $\forall E_k \in \mathscr{E}_1^{\mathbb{V}_q}$, $\exists\,E_h \in \mathscr{E}_2^{\mathbb{V}_q}$ such that $E_k \subseteq E_h$.*

We write $\vee_{\mathbb{E}}$ and $\wedge_{\mathbb{E}}$ to refer to the least upper bound (lub) and the greatest lower bound

(glb) induced by $\leqslant_{\mathbb{E}}$, and the resulting domain is a complete lattice. For instance,

$$[(\{a,b,c\},0),(\{d\},1)] \leqslant_{\mathbb{E}} [(\{a,b\},0),(\{c\},0),(\{d\},1)] \leqslant_{\mathbb{E}}$$
$$\leqslant_{\mathbb{E}} [(\{a\},0),(\{b\},0),(\{c\},0),(\{d\},1)] \leqslant_{\mathbb{E}} [(\{a\},0),(\{b\},0),(\{c\},0),(\{d\},0)] \text{ and}$$
$$[(\{a,b\},0),(\{c\},1)] \vee_{\mathbb{E}} [(\{a\},0),(\{b,c\},1)] = [(\{a\},0),(\{b\},0),(\{c\},0)]$$

To ensure soundness, we overestimate the non-separabilities and determine which variables are potentially inseparable. On the other hand, since being d-inseparable implies special effects in relation to control-not and measurement, we underestimate the d-inseparability property to make sure we do not introduce errors in the abstract semantics.

We have defined an abstract domain that allows us to represent inseparability and d-inseparability. However, we need a final ingredient to define the abstract semantics: some elements abstracting the variables' state.

Labels

We introduce some labels that represent some specific states that are relevant to entanglement abstraction. The CX gate does not introduce entanglement if the controller is in a classical state ($|0\rangle$ or $|1\rangle$) or the target is in a uniform superposition ($1/\sqrt{2}\,|0\rangle \pm |1\rangle$). To track these two states, we introduce two labels that represent two sets of states: $Z = \{\phi\,|b\rangle\}$ (the set of classical values), and $X = \{\phi(1/\sqrt{2}\,|b\rangle \pm 1/\sqrt{2}\,|\bar{b}\rangle)\}$ (the set of values in uniform superposition), where ϕ represents a global phase, b is a binary string and $\bar{b}$ is the negation of b (e.g. if $b = 0$ then $\bar{b} = 1$ and if $b = 010$ then $\bar{b} = 101$). Moreover, we introduce three other labels:

$$P = \{\phi(1/\sqrt{2}\,|b\rangle \pm i + 1/\sqrt{2}\,|\bar{b}\rangle)\} \qquad Y = \{\phi(1/\sqrt{2}\,|b\rangle \pm i/\sqrt{2}\,|\bar{b}\rangle)\}$$
$$R = \{\phi(1/\sqrt{2}\,|b\rangle \pm i - 1/\sqrt{2}\,|\bar{b}\rangle)\}.$$

We need these labels to represent the semantics of the gate $\mathtt{t}$. In particular, $\mathtt{t}X = P$, $\mathtt{t}P = Y$, $\mathtt{t}Y = R$ and $\mathtt{t}R = X$ while $\mathtt{t}Z = Z$. Then, we add a final label to represent states that are not classical, i.e., those that are definitely in superposition: $S = \{\phi(\alpha\,|b\rangle \pm \beta\,|\bar{b}\rangle) \mid |\alpha|^2 + |\beta|^2 = 1 \text{ and } \alpha \neq 0 \wedge \beta \neq 0\}$. Finally, we define $\bot_{\mathscr{L}}$ as the empty set and $\top_{\mathscr{L}}$ as the set of all possible vectors. We can order these labels by inclusion, constructing a lattice $(\mathscr{L}, \leqslant_{\mathscr{L}})$, represented in Figure 5.1.

5.3.1 Put all together: The Abstract Domain

Now, we include the labels in the definition of the abstract domain. When variables are inseparable, their state cannot be described as a combination of individual states. Recall

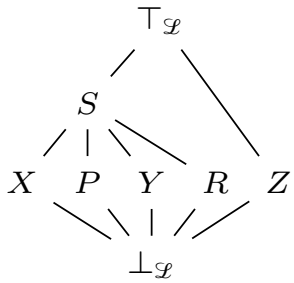


Figure 5.1: Lattice $(\mathscr{L}, \leqslant_{\mathscr{L}})$

that, in an abstract state $\mathscr{C}^{\mathbb{V}_q}$, a set of inseparable variables is the union of all sets e with the same index k. We introduce a labelling function to associate each set of inseparable variables with a label. Formally, we define the labelling function as $\lambda : \mathbb{N} \to \mathscr{L}$, and we call $\Lambda = \mathbb{N} \times \mathscr{L}$ the domain of the labelling function. Consequently, we reformulate the definition of the abstract state, including the labelling function, as follows.

Definition 5.5. *Given a set of quantum variables $\mathbb{V}_q$, let be $\mathscr{C}^{\mathbb{V}_q} \in \mathbb{E}^{\mathbb{V}_q}$ and $\lambda \in \Lambda$, we define an abstract state as the pair $(\mathscr{C}^{\mathbb{V}_q}, \lambda)$.*

We define $\mathbb{A}^{\mathbb{V}_q} = \mathbb{E}^{\mathbb{V}_q} \times \Lambda$ as the abstract domains. For instance, let us consider a set of variables $Q = \{a, b, c, d\}$ in the state

$$|\psi\rangle_Q = (1/2(|0\rangle + |1\rangle)\,|0\rangle + i/2(|0\rangle - |1\rangle)\,|1\rangle)_{a,b} \otimes 1/\sqrt{2}(|10\rangle + |01\rangle)_{c,d}.$$

First, we construct the partition corresponding to the sets of inseparable variables, i.e., the sets $\{a, b\}$ and $\{c, d\}$. Once these two sets have been identified, we construct the abstract state by identifying which variables are d-inseparable, obtaining the abstract state: $[(\{a\}, 0)(\{b\}, 0)(\{d, c\}, 1)]$. The last step is to label the sets of inseparable variables. In particular, given a set of variables in a state $|\phi\rangle$, we choose the smallest label L such that $|\phi\rangle \in L$. We see that the state of the variables a, b is contained only in $\top_{\mathscr{L}}$ while the state of c, d is contained in $X, S, \top_{\mathscr{L}}$, consequently the final state will be equal to:

$$(\mathscr{C}^{\mathbb{V}_q}, \lambda) = ([(\{a\}, 0)(\{b\}, 0)(\{c, d\}, 1)], \{0 : \top_{\mathscr{L}}, 1 : X\}.$$

At this point, if we want to compute the set of concrete states represented by an abstract state, starting from the obtain $(\mathscr{C}^{\mathbb{V}_q}, \lambda)$, we consider $\mathscr{C}^{\mathbb{V}_q}$. In this case, we know that there are two sets $\{a, b\}$, $\{c, d\}$ of inseparable variables, so the abstract state corresponds to a set of concrete states in the form $|\phi_1\rangle_{a,b} \otimes |\phi_2\rangle_{c,d}$. Then, we can get more precise information about $|\phi_1\rangle$ and $|\phi_2\rangle$ by checking which variables are d-inseparable, that is,

c and d. In particular, we know that the concrete states can be expressed by the set $\{|\Psi\rangle \mid |\Psi\rangle = (|\phi_1\rangle_{a,b} \otimes (|b\rangle + |\bar{b}\rangle)_{c,d}\}$. Now, we complete the concretisation by checking the information contained in the labels. By the labels we know that a, b can be in any quantum state while c, d are in the form of $1/\sqrt{2}(|b\rangle + |\bar{b}\rangle)_{c,d}$. Finally, by intersecting this information with the previous one, we obtain the set: $\{|\psi\rangle \mid |\psi\rangle = (|\phi_1\rangle_{a,b} \otimes 1/\sqrt{2}(\alpha\,|b\rangle + \beta\,|\bar{b}\rangle)_{c,d}\}$, which represents the concretisation $\gamma((\mathscr{E}^{\mathbb{V}_q}, \lambda))$ of $(\mathscr{E}^{\mathbb{V}_q}, \lambda)$ and, of course, $|\psi\rangle \in \gamma((\mathscr{E}^{\mathbb{V}_q}, \lambda))$. Note that labels represent the state of a single variable or a state of n d-inseparable variables. So, if a set of inseparable variables is not labelled as $\top_{\mathscr{L}}$, Z, or $\bot_{\mathscr{L}}$, it means that it corresponds either to a single variable or to all d-inseparable variables.

Based on the partial order $\leqslant_{\mathbb{E}}$ defined in Definition 5.4, we can define an ordering in $\mathbb{A}^{\mathbb{V}_q}$.

Definition 5.6 $(\mathbb{A}^{\mathbb{V}_q}, \sqsubseteq)$**.** *Given* $(\mathscr{E}_1^{\mathbb{V}_q}, \lambda_1), (\mathscr{E}_2^{\mathbb{V}_q}, \lambda_2) \in \mathbb{A}^{\mathbb{V}_q}$, $(\mathscr{E}_1^{\mathbb{V}_q}, \lambda_1) \sqsubseteq (\mathscr{E}_2^{\mathbb{V}_q}, \lambda_2)$ *if and only if*

$$\mathscr{E}_1^{\mathbb{V}_q} \leqslant_{\mathbb{E}} \mathscr{E}_2^{\mathbb{V}_q} \;\wedge\; \forall E_h \in \mathscr{E}_2, \begin{cases} \lambda(k) \leqslant_{\mathscr{L}} \lambda(h) & \text{if } \exists E_k \in \mathscr{E}_1 \text{ such that } E_h = E_k \\ \lambda(h) = \top_{\mathscr{L}} & \text{otherwise} \end{cases}$$

We call $\sqcup$ and $\sqcap$ the lub and glb induced by the order. Since the lattices $(\mathbb{E}^{\mathbb{V}_q}, \leqslant_{\mathbb{E}})$ and $(\mathscr{L}, \leqslant_{\mathscr{L}})$ are finite and defined by inclusion operators, they are complete lattices. Thereby, also the lattice $((\mathscr{E}^{\mathbb{V}_q}, \lambda), \sqsubseteq)$ is complete.

For instance, consider two abstract states $(\mathscr{E}_1^{\mathbb{V}_q}, \lambda_1) = ([(\{p, q\}, 0)], 0 : X)$ and $(\mathscr{E}_2^{\mathbb{V}_q}, \lambda_2) = ([(\{p, q\}, 0)], 0 : Y)$; the lub between them is $(\mathscr{E}_3^{\mathbb{V}_q}, \lambda_3) = ([(\{p, q\}, 0)],\ 0 : S)$. In this case, since p, q are d-inseparable in both states, they are also in the lub, and we label the partition by the lub between the labels (we are in the 'if' case of the Definition 5.6). In fact, $(\mathscr{E}_1^{\mathbb{V}_q}, \lambda_1)$ represent the set of states $\{|\psi\rangle \mid \phi(1/\sqrt{2}\,|b\rangle \pm 1/\sqrt{2}\,|\bar{b}\rangle)\}$ and $(\mathscr{E}_2^{\mathbb{V}_q}, \lambda_2)$ represent the set of states $\{|\psi\rangle \mid \phi(1/\sqrt{2}\,|b\rangle \pm i/\sqrt{2}\,|\bar{b}\rangle)\}$, where $b \in \{00, 01, 10, 11\}$, and the abstraction of the union of these two sets is $(\mathscr{E}_3^{\mathbb{V}_q}, \lambda_3)$.

On the other hand, if we consider the states

$$(\mathscr{E}_4^{\mathbb{V}_q}, \lambda_4) = ([(\{p, q\}, 0), (\{t\}, 1)], \{0 : X, 1 : X\})$$

and

$$(\mathscr{E}_5^{\mathbb{V}_q}, \lambda_5) = ([(\{p\}, 0), (\{q, t\}, 1)], \{0 : X, 1 : X\},$$

the lub between them is

$$(\mathscr{E}_6^{\mathbb{V}_q}, \lambda_6) = ([(\{p\}, 0), (\{q\}, 0), (\{t\}, 0)], \{0 : \top_{\mathscr{L}}\}.$$

In fact, $(\mathfrak{C}_4^{\mathbb{V}_q}, \lambda_4)$ represent the set of states

$$\{|\psi\rangle \mid \phi(1/\sqrt{2}\,|b\rangle \pm 1/\sqrt{2}\,|\bar{b}\rangle)_{p,q} \otimes 1/\sqrt{2}\,|0\rangle \pm 1/\sqrt{2}\,|1\rangle)_t\}$$

and $(\mathfrak{C}_5^{\mathbb{V}_q}, \lambda_5)$ represent the set of states

$$\{|\psi\rangle \mid \phi(1/\sqrt{2}\,|0\rangle \pm 1/\sqrt{2}\,|1\rangle)_p \otimes 1/\sqrt{2}\,|b\rangle \pm 1/\sqrt{2}\,|\bar{b}\rangle)_{q,t}\}.$$

If we join these sets, we obtain a set of states where p, q, t are inseparable. However, p and q are only d-inseparable in some states, while q and t are d-inseparable in others. So, in general, we can only say that p, q, t are inseparable and not d-inseparable, and the only possible label for these states is $\top_{\mathscr{L}}$.

Concretisation Function

Now, we can introduce some formalism to define the concretisation function $\gamma : \mathbb{A} \to \wp(V_{\mathbb{V}_q})$.

Definition 5.7. *Let $\mathbb{V}_q$ be a set of variables, and $\pi \subset \wp(\mathbb{V}_q)$ a partition of $\mathbb{V}_q$. Given a state $|\psi\rangle_{\mathbb{V}_q}$, we say that the variables in $\mathbb{V}_q$ are π-separable if $|\psi\rangle_{\mathbb{V}_q} = \bigotimes_{p\in\pi} |\phi\rangle_p$, i.e., their joint state can be decomposed into a product of states across a partition π of the variables.*

For instance, given a state $|\psi\rangle_{q_1,q_2,q_3}$ and a partition $\pi = \{\{q_1, q_2\}\{q_3\}\}$, q_1, q_2, q_3 are π-separable if and only if we can write $|\psi\rangle_{q_1,q_2,q_3}$ as $|\phi_1\rangle_{q_1,q_2} \otimes |\phi_2\rangle_{q_3}$.

Given an abstract state $(\mathfrak{C}^{\mathbb{V}_q}, \lambda)$, we write $\{E_k\}$ to indicate the sets that are obtained by $\mathfrak{C}^{\mathbb{V}_q} = \{(e, k)\}$ joining the sets e with the same k. Recall that $\{E_k\}$ is a partition that represents the sets of inseparable variables.

Definition 5.8. *Given a set of variables $\mathbb{V}_q$, we say that $|\psi\rangle_{\mathbb{V}_q} \triangleright (\mathfrak{C}^{\mathbb{V}_q}, \lambda)$ (that is, $|\psi\rangle_{\mathbb{V}_q}$ is abstracted by $(\mathfrak{C}^{\mathbb{V}_q}, \lambda)$) if and only if*

- *$|\psi\rangle_{\mathbb{V}_q}$ is $\{E_k\}$-separable;*
- *$\forall (e, k) \in \mathfrak{C}^{\mathbb{V}_q}$, $q_i, q_j \in e \Rightarrow q_j$ and q_i are d-inseparable in $|\psi\rangle_{\mathbb{V}_q}$;*
- *given $|\psi\rangle_{\mathbb{V}_q} = \bigotimes_k |\psi\rangle_{E_k}$, for all k, $|\psi\rangle_{E_k} \in \lambda(k)$.*

In other words, we say that an abstract state $\mathfrak{C}^{\mathbb{V}_q}$ abstracts a concrete state if and only if the abstract state over-approximates the set of inseparable variables, under-approximates the d-inseparability, and all separable sub-states that compose the state are represented by labels. Now we have all the elements to define the concretisation function $\gamma : \mathbb{A} \to \wp(V_{\mathbb{V}_q})$:

$$\gamma(\mathfrak{C}^{\mathbb{V}_q}, \lambda) = \{|\psi\rangle_{\mathbb{V}_q} \mid |\psi\rangle_{\mathbb{V}_q} \triangleright (\mathfrak{C}^{\mathbb{V}_q}, \lambda)\}.$$

Theorem 5.1. *Given a set of abstract states* $\{(\mathscr{E}_j^{\mathbb{V}_q}, \lambda_j)\}_j$*, let* $I = \bigcap_j \gamma(\mathscr{E}_j^{\mathbb{V}_q}, \lambda_j)$*, exists an abstract state* $(\mathscr{E}_I^{\mathbb{V}_q}, \lambda_I)$ *such that* $\gamma((\mathscr{E}_I^{\mathbb{V}_q}, \lambda_I)) = I$*.*

Sketch. $\gamma(\mathscr{E}_j^{\mathbb{V}_q}, \lambda_j)$ is the set of state in $\mathscr{H}_{\mathbb{V}_q}$ that are abstracted by $(\mathscr{E}_j^{\mathbb{V}_q}, \lambda_j)$, so I is the set of states that are abstracted by all $(\mathscr{E}_j^{\mathbb{V}_q}, \lambda_j)$. Consequently, if $(\mathscr{E}_I^{\mathbb{V}_q}, \lambda_I) = \bigsqcap_j (\mathscr{E}_j^{\mathbb{V}_q}, \lambda_j)$ then $\gamma((\mathscr{E}_I^{\mathbb{V}_q}, \lambda_I)) = I$. □

Theorem 5.1 proves that $\mathbb{A}$ is isomorphic to a Moore family of $\wp(V_Q)$. This means that exist a function $\alpha_l : \wp(V_{\mathbb{V}_q}) \to \mathbb{A}$ such that $\langle \mathbb{A}, \alpha_l, \gamma_l, \wp(V_{\mathbb{V}_q}) \rangle$ forms a Galois Insertion [War42; CC79; CC77].

5.4 Abstract Semantics

In this section, we define the abstract semantics for our analysis. We first need to introduce some additional notation to better represent the operations on abstract states. Let us consider a generic abstract state $(\mathscr{E}^{\mathbb{V}_q}, \lambda)$, where $\mathscr{E}^{\mathbb{V}_q} = \{(e_i, k_i)\}_i$. We write $\mathscr{E}^{\mathbb{V}_q}(q)$ to refer to the pair $(e, k) \in \mathscr{E}^{\mathbb{V}_q}$ such that $q \in e$, while we have $\mathscr{E}^{\mathbb{V}_q}\{q\}$ to refer to set $E_k \in \mathscr{E}^{\mathbb{V}_q}$ that contains q. For instance, if $\mathscr{E}_1 = [(\{q\}, 0), (\{t\}, 0), (\{r\}, 1)]$ then $\mathscr{E}_1(q) = (\{q\}, 0)$ and $\mathscr{E}_1\{q\} = \{q, t\}$. We write $\mathscr{E}^{\mathbb{V}_q}[q_1 + q_2]$ to mark q_1 and q_2 inseparable in $\mathscr{E}^{\mathbb{V}_q}$ (e.g., $\mathscr{E}_1[r + q]$ is equal to $[(\{q\}, 1), (\{t\}, 1), (\{r\}, 1)]$). Note that if we mark q and r as inseparable, this also affects t due to the transitivity of the inseparability. $\mathscr{E}^{\mathbb{V}_q}[q_1 \uplus q_2]$ means that we marked q_1 and q_2 as d-inseparable, so given $\mathscr{E}_1$ from above, $\mathscr{E}_1[q \uplus t] = [(\{q, t\}, 0), (\{r\}, 1)]$. Note that $\mathscr{E}[q_1 \uplus q_2]$ implies applying also $\mathscr{E}[q_1 + q_2]$. $\mathscr{E}^{\mathbb{V}_q}[\sim q]$ denotes the removal of q from the set of variables d-inseparable from itself, and $\mathscr{E}^{\mathbb{V}_q}[\neg q]$ denotes that we make q separable from the rest of the variables. For instance, given $\mathscr{E}_2 = [(\{q, r\}, 0), (\{t\}, 0)]$, $\mathscr{E}_2[\sim q]$ is equal to $[(\{r\}, 0), (\{q\}, 0), (\{t\}, 0)]$ and $\mathscr{E}_2[\neg t]$ correspond to $[(\{q, r\}, 0), (\{t\}, 1)]$. Of course, $\mathscr{E}_2[\neg q_1]$ implies $\mathscr{E}_2[\sim q_1]$. Finally, to ease the use of the labelling function, given a variable q, let $\mathscr{E}^{\mathbb{V}_q}(q) = (e, k)$, we write $\lambda(q)$ to refer to $\lambda(k)$. Additionally, we write $\lambda[q \leftarrow L]$ to state that we change the label referred to the index k associated with q, setting it equal to L. For instance, given $(\mathscr{E}^{q,p,t}, \lambda) = ([(\{q\}, 0)(\{p\}, 0), (\{t\}, 1)], \{0 : \top_{\mathscr{L}}, 1 : X\})$, $\lambda(q) = \lambda(p)$ correspond to $\lambda(0) = \top_{\mathscr{L}}$, $\lambda(t) = X$, since it corresponds to $\lambda(1)$, and $(\mathscr{E}^{q,p,t}, \lambda[q \leftarrow Y])$ is equal to $([(\{q\}, 0)(\{p\}, 0), (\{t\}, 1)], \{0 : Y, 1 : X\})$. In general, when we speak about a label associated with a variable q, we implicitly refer to the label associated with the index associated with q.

Before defining the abstract semantics of the language, we define four abstract operations $H_q^\sharp, T_q^\sharp, CX_{p,q}^\sharp, M_q^\sharp : \mathbb{A}^{\mathbb{V}_q} \to \mathbb{A}^{\mathbb{V}_q}$, that correspond to the abstraction of the unitary operation H, T, CX and the measurement, respectively.

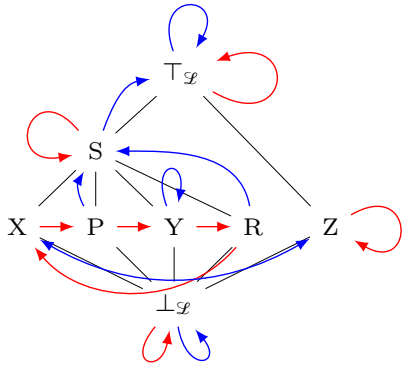


Figure 5.2: The red arrow indicates the semantics of the abstract operation $T_q^\sharp$ while the blue one indicates the semantics of the abstract operation $H_q^\sharp$.

For all the abstract operations it holds that if $(\mathscr{E}^{\mathbb{V}_q}, \lambda) = \bot$, then $G_v^\sharp(\mathscr{E}^{\mathbb{V}_q}, \lambda) = (\mathscr{E}^{\mathbb{V}_q}, \lambda)$. If fact $\gamma(\bot) = \varnothing$ and $G_v \varnothing = \varnothing$.

T gate ($T_q^\sharp$)

This gate does not make a difference if we apply it to a single or a group of d-inseparable variables. Formally,

$$T_q^\sharp(\mathscr{E}^{\mathbb{V}_q}, \lambda) = (\mathscr{E}^{\mathbb{V}_q}, \lambda[q \leftarrow V]),$$

where V can be derived from the red arrows in Figure 5.2.

Hadamard gate ($H_q^\sharp$)

The Hadamard gate distinguishes whether q is entangled with other variables. Formally, the abstract semantics is defined as follows:

$$H_q^\sharp(\mathscr{E}^{\mathbb{V}_q}, \lambda) = \begin{cases} (\mathscr{E}^{\mathbb{V}_q}, \lambda[q \leftarrow V]) & |\mathscr{E}^Q\{q\}| = 1, \\ (\mathscr{E}^{\mathbb{V}_q}(\sim q), \lambda[q \leftarrow \top_{\mathscr{L}}]) & \text{otherwise.} \end{cases}$$

In particular, if q is separable from the other variables (i.e., $|\mathscr{E}^Q\{q\}| = 1$), V can be derived from the blue arrows in Figure 5.2. If q is inseparable (i.e., $|\mathscr{E}^Q\{q\}| > 1$), applying Hadamard to it produces a state that we can only label with $\top_{\mathscr{L}}$, and the variable q is no longer d-inseparable. For instance, given $|\psi\rangle_{p,q,t} = (1/\sqrt{2}(|000\rangle + |111\rangle)_{p,q,t}$, p, q, t are d-inseparable and their state can be labelled as X. Then, applying $H_q |\psi\rangle_{p,q,t} = (|000\rangle + |010\rangle + |101\rangle + |111\rangle)_{p,q,t}$, q is no longer d-inseparable from p, t, and the state is only labelable by $\top_{\mathscr{L}}$.

Controlled-Not gate ($CX^\sharp_{c,t}$)

The CX gate can introduce or nullify entanglement, so we need to consider different cases according to the state of c and t. Given an abstract state $(\mathfrak{E}^{\mathbb{V}_q}, \lambda)$, we can define the abstract semantics as follows:

- if $\lambda(c) = Z$, i.e., the controller is in a base value, the CX corresponds to a classical controlled not, so it does not introduce or nullify entanglement and it does not change labels, thus:
$$CX^\sharp_{c,t}(\mathfrak{E}^{\mathbb{V}_q}, \lambda) = (\mathfrak{E}^{\mathbb{V}_q}, \lambda);$$
- if t is separable from other variables (i.e. $|\mathfrak{E}^{\mathbb{V}_q}\{t\}| = 1$) we check the state of c and t:
$$CX^\sharp_{c,t}(\mathfrak{E}^{\mathbb{V}_q}, \lambda) = \begin{cases} (\mathfrak{E}^{\mathbb{V}_q}, \lambda) & \text{if } \lambda(t) = X, \\ (\mathfrak{E}[c \uplus t], \lambda) & \text{if } \lambda(c) \neq \top_{\mathscr{L}} \wedge \lambda(t) = Z, \\ ((\mathfrak{E}[\sim t])[c + t], \lambda[t \leftarrow \top_{\mathscr{L}}]) & \text{otherwise.} \end{cases}$$
In particular, if the target is in uniform superposition, the CX gate corresponds to the identity.
If the controller is surely in superposition (i.e., $\lambda(c) \neq \top_{\mathscr{L}}$ and $\lambda(c) \neq Z$) and the target is a classical value, the CX gate makes t d-inseparable from c. Moreover, when we set t d-inseparable from c, t automatically gets the label of c. For instance, if we have three variables, q, c, t in the state $|\psi_1\rangle = {}^1/\sqrt{2}(|00\rangle + |11\rangle)_{q,c} \otimes |1\rangle_t$, applying $CX_{c,t}\,|\psi_1\rangle$ we obtain ${}^1/\sqrt{2}(|001\rangle + |110\rangle)_{q,c,t}$ in which q, c, t are d-inseparable. Then, given $Q = \{q, c, t\}$ and $(\mathfrak{E}_1^{\mathbb{V}_q}, \lambda_1) = \{(\{q, c\}, 0), (\{t\}, 1)\}, \{0 : X, 1 : Z\})$ such that $|\psi_1\rangle \in \gamma(\mathfrak{E}_1^{\mathbb{V}_q}, \lambda_1)$, $CX^\sharp_{c,t}(\mathfrak{E}_1^{\mathbb{V}_q}, \lambda_1) = (\{(\{q, c, t\}, 0)\}, \{0 : X\})$ and $CX_{c,t}\,|\psi_1\rangle \in \gamma(CX^\sharp_{c,t}(\mathfrak{E}_1^{\mathbb{V}_q}, \lambda_1))$.
Finally, if none of the above conditions is fulfilled, we need to approximate the relation between c and t. To maintain the soundness, we mark c and t as inseparable without setting c and t d-inseparable. Moreover, we mark c and t equal to $\top_{\mathscr{L}}$ (recall that, since t and c are inseparable, they are related to the same k, so writing $\lambda[t \leftarrow \top_{\mathscr{L}}]$ or $\lambda[c \leftarrow \top_{\mathscr{L}}]$ produce the same effects).
- if t is inseparable from other variables, we need to check if the CX gate nullifies some entanglements:
$$CX^\sharp_{c,t}(\mathfrak{E}^{\mathbb{V}_q}, \lambda) = \begin{cases} (\mathfrak{E}[\neg t], \lambda[t \leftarrow Z]) & \mathfrak{E}^{\mathbb{V}_q}(c) = \mathfrak{E}^{\mathbb{V}_q}(t) \\ (\mathfrak{E}[\sim t], \lambda[t \leftarrow \top_{\mathscr{L}}]) & \text{otherwise} \end{cases}.$$
In particular, if c and t are d-inseparable ($\mathfrak{E}^{\mathbb{V}_q}(c) = \mathfrak{E}^{\mathbb{V}_q}(t)$), we 'disentangle' t, as we see in Equation 5.3, and, we set the disentangled variable separable from the rest, labelling it

as Z. Otherwise, we do not know the exact effect of the gate since, as we have shown in Equation 5.4, the CX alters the entangled state. Thus, to maintain soundness, we mark t as not d-inseparable from the other variables and label it as $\top_{\mathscr{L}}$. For instance, given $(\mathscr{E}_1^{\mathbb{V}_q}, \lambda_1) = ([(\{a,b\},0),(\{c\},0)],\{0:\top_{\mathscr{L}}\})$, $CX^\sharp_{a,b}(\mathscr{E}_1^{\mathbb{V}_q}, \lambda_1) = ([(\{a\},0),(\{c\},0),(\{b\},1)],\{0:\top_{\mathscr{L}}, 1:Z\})$ while $CX^\sharp_{c,a}(\mathscr{E}_1^{\mathbb{V}_q}, \lambda_1) = ([(\{a\},0),\ (\{c\},0),(\{b\},0)],\{0:\top_{\mathscr{L}}\})$. Given $|\psi\rangle_{a,b,c}$, $CX_{a,b}\,|\psi\rangle_{a,b,c}$ and $CX_{c,a}\,|\psi\rangle_{a,b,c}$ from Equation 5.3 and Equation 5.4, note that $|\psi\rangle_{a,b,c} \in \gamma((\mathscr{E}_1^{\mathbb{V}_q}, \lambda_1))$, $CX_{a,b}\,|\psi\rangle_{a,b,c} \in \gamma(CX^\sharp_{c,a}(\mathscr{E}_1^{\mathbb{V}_q}, \lambda_1))$ and $CX_{c,a}\,|\psi\rangle_{a,b,c} \in \gamma(CX^\sharp_{a,b}(\mathscr{E}_1^{\mathbb{V}_q}, \lambda_1))$.

Measurement $(M_q^\sharp)$

In this case, we need to approximate which variables may be affected by the measurement. The semantics of measurement is formally defined as:

$$M_q^\sharp(\mathscr{E}^{\mathbb{V}_q}, \lambda) = (\mathscr{E}[\neg Q], L[Q \leftarrow Z, T \leftarrow \top_{\mathscr{L}}]),$$

where $Q = \mathscr{E}^{\mathbb{V}_q}(q)$ and $T = (\mathscr{E}^{\mathbb{V}_q}\{q\} \setminus Q)$. In particular, when a variable q is separable, $Q = \{q\}$ and $T = \varnothing$, and the measurement simply makes q collapse to a base state. Instead, if q is inseparable from other variables, all variables d-inseparable from q collapse to Z (making them separable), while all other variables inseparable and not d-inseparable from q are altered in a way that cannot be modelled given an abstract state. For this reason, we can only label these variables with $\top_{\mathscr{L}}$.

Language Abstract Semantics

Now we have all the ingredients to define the abstract semantics of the language, which we define as a function $(\!|\cdot|\!) : \mathbb{A}^{\mathbb{V}_q} \to \mathbb{A}^{\mathbb{V}_q}$, for each instruction of our language:

$$\begin{aligned}
(\!| \mathit{skip} |\!)(\mathscr{E}^{\mathbb{V}_q}, \lambda) &= (\mathscr{E}^{\mathbb{V}_q}, \lambda) \\
(\!| \mathtt{h}(q) |\!)(\mathscr{E}^{\mathbb{V}_q}, \lambda) &= H_q^\sharp(\mathscr{E}^{\mathbb{V}_q}, \lambda) \\
(\!| \mathtt{t}(q) |\!)(\mathscr{E}^{\mathbb{V}_q}, \lambda) &= T_q^\sharp(\mathscr{E}^{\mathbb{V}_q}, \lambda) \\
(\!| \mathtt{cx}(p,q) |\!)(\mathscr{E}^{\mathbb{V}_q}, \lambda) &= CX_{p,q}^\sharp(\mathscr{E}^{\mathbb{V}_q}, \lambda) \\
(\!| \mathtt{M}_1(b) |\!)(\mathscr{E}^{\mathbb{V}_q}, \lambda) &= M_q^\sharp(\mathscr{E}^{\mathbb{V}_q}, \lambda) \\
(\!| \mathtt{M}_0(b) |\!)(\mathscr{E}^{\mathbb{V}_q}, \lambda) &= M_q^\sharp(\mathscr{E}^{\mathbb{V}_q}, \lambda) \\
(\!| \mathtt{c}_1; \mathtt{c}_1 |\!)(\mathscr{E}^{\mathbb{V}_q}, \lambda) &= (\!| \mathtt{c}_2 |\!)((\!| \mathtt{c}_1 |\!)(\mathscr{E}^{\mathbb{V}_q}, \lambda)) \\
(\!| \mathtt{c}_1 \oplus \mathtt{c}_2 |\!)(\mathscr{E}^{\mathbb{V}_q}, \lambda) &= (\!| \mathtt{c}_1 |\!) \sqcup (\!| \mathtt{c}_2 |\!) \\
(\!| \mathtt{c}^* |\!)(\mathscr{E}^{\mathbb{V}_q}, \lambda) &= \bigsqcup_n (\!| \mathtt{c}^n |\!)(\mathscr{E}^{\mathbb{V}_q}, \lambda).
\end{aligned} \tag{5.5}$$

where $H_q^\sharp, T_q^\sharp, CX_{p,q}^\sharp, M_q^\sharp : \mathbb{A}^{\mathbb{V}_q} \to \mathbb{A}^{\mathbb{V}_q}$ represent the abstract semantics of gates and measurement.

Finally, we formulate the soundness of our abstraction in terms of the concretisation function γ [CC79].

Proposition 5.3 (Soundness)**.** *Let $\mathbb{V}_q$ be a set of variables, $\forall l \in$ label, $\forall(\mathscr{E}^{\mathbb{V}_q}, \lambda) \in \mathbb{A}$, $[\![l]\!] \circ \gamma(\mathscr{E}^{\mathbb{V}_q}, \lambda) \subseteq \gamma \circ (\!| l |\!)(\mathscr{E}^{\mathbb{V}_q}, \lambda)$.*

Evidence in support of this proposition is shown by the examples in Proposition 5.5. Since every label is sound, by induction on c we can prove that $[\![\mathsf{c}]\!] \circ \gamma(\mathscr{E}^{\mathbb{V}_q}, \lambda) \subseteq \gamma \circ (\!| \mathsf{c} |\!)(\mathscr{E}^{\mathbb{V}_q}, \lambda)$.

5.5 Computing the Analysis

To compute the analysis on the CFG, we need to compute the abstract state $(\mathscr{E}^{\mathbb{V}_q}, \lambda)$ for each node of the CFG, namely at each program point of the analysed program [SWH12]. The analysis we propose is forward; namely, the state $(\mathscr{E}^{\mathbb{V}_q}, \lambda)$ at node u, denoted $(\mathscr{E}^{\mathbb{V}_q}, \lambda)[u]$, depends on the pairs $\{(\mathscr{E}_i^{\mathbb{V}_q}, \lambda_i)\}$ of its predecessors and the label semantics of the edges entering u. Given a CFG G, for all node u in G (program points of the represented program), we define the following system of equations:

$$(\mathscr{E}^{\mathbb{V}_q}, \lambda)[u] = \begin{cases} (\mathscr{E}_0^{\mathbb{V}_q}, \lambda_0) & \text{if } u = \textbf{start} \\ \bigsqcup \{\ (\!| l |\!)(\mathscr{E}^{\mathbb{V}_q}, \lambda)[u] \mid (u, l, v) \in G \} & \text{otherwise} \end{cases} \tag{5.6}$$

where $(\mathscr{E}_0^{\mathbb{V}_q}, \lambda_0)$ is the initial state. Since we assume that all variables are initialised to $|0\rangle$, the initial state is the state where all variables are separable and labelled as Z. So if $\mathbb{V}_q = \{a, b, c\}$ then $(\mathscr{E}_0^{\mathbb{V}_q}, \lambda_0) = ([(\{a\},0), (\{b\},1), (\{c\},2)], \{0,1,2,:Z\})$.

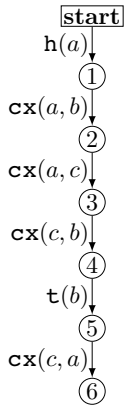
This system can be solved by the least fixed point obtaining the best solution for each program point [KSS09].

Proposition 5.4. *For all statement $l \in$ label, the abstract semantics $(\!| l |\!) : \mathbb{A} \to \mathbb{A}$ is monotonic w.r.t. $\sqsubseteq$.*

Since the semantics is monotonic, it is granted that we reach the fix-point.

We provide a prototype of our procedure* that analyses the quantum language used to present the analysis. Together with the prototype, we provide examples showing how our analysis works in various scenarios. In particular, we analyse the examples contained

* The following GitHub repository https://github.com/NicolaAssolini98/EntaglementAnalysis contains our prototype implemented in Python



(a)

	Concrete state	Abstract state
Start	$\{\|000\rangle_{a,b,c}\}$	[({a},0),({b},1),({c},2)],{(0,1,2):Z})
1	$\{^1/\sqrt{2}(\|0\rangle + \|1\rangle)_a \|00\rangle_{b,c}\}$	[({a},0),({b},1),({c},2)],{0:X,(1,2):Z})
2	$\{^1/\sqrt{2}(\|00\rangle + \|11\rangle)_{a,b} \|0\rangle_c\}$	[({a,b},0),({c},2)],{0:X,2:Z})
3	$\{^1/\sqrt{2}(\|000\rangle + \|111\rangle)_{a,b,c}\}$	[({a,b,c},0)],{0:X})
4	$\{^1/\sqrt{2}(\|00\rangle + \|11\rangle)_{a,c} \|0\rangle_b\}$	[({a,c},0),({b},1)],{0:X,1:Z})
5	$\{^{i+1}/\sqrt{2}(\|00\rangle + \|11\rangle)_{a,c} \|0\rangle_b\}$	[({a,c},0),({b},1)],{0:P,1:Z})
6	$\{^{i+1}/\sqrt{2}(\|0\rangle + \|1\rangle)_c \|0\rangle_b \|0\rangle_c\}$	[({c},0),({b},1),({a},2)],{0:P,(1,2):Z})

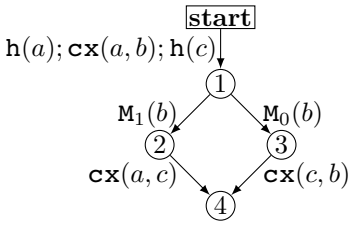
(b)

Figure 5.3: The $dGHZ$ CFG (a), and a table representing concrete and abstract states for each node (b).

in [Ran+20](superdense coding [BW92], Deutsch algorithm [DP85], and the Creation and disentanglement of the GHZ state) and in [Per08b] (teleportation circuit and GHZ), obtaining the same results as [Ran+20] and improving [Per08b]. We also provide some examples showing how we lose precision in the presence of control flow, showing when our analysis is sound but incomplete.

Showing the Analysis

Consider the example in Figure 5.3. In Figure 5.3, we show the CFG to the program $dGHZ ::= \texttt{h}(a); \texttt{cx}(a,b); \texttt{cx}(a,c); \texttt{cx}(c,b); \texttt{t}(b); \texttt{cx}(c,a)$ displaying for each program point the concrete state in blue and the abstract state in black. This program creates the GHZ states up to node **3**, then 'disentangles' b with the first `cx`, then changes the relative phase with the `t` gate, and then disentangles c with the last `cx`. In the abstract domain up to node **3**, we construct the state in which $\{a,b,c\}$ are d-inseparable. Then we are able to keep track in the abstract state the entanglement cancellation made by the `cx` gate in edges $(\mathbf{3}, \texttt{cx}(a,b), \mathbf{4})$ and $(\mathbf{5}, \texttt{cx}(c,a), \mathbf{6})$ and the phase change made by the `t` gate in edge $(\mathbf{4}, \texttt{t}(b), \mathbf{5})$. We show how our analysis works with control flow in Figure 5.4. We consider the program $prog ::= \texttt{h}(a); \texttt{cx}(a,b); \texttt{h}(c); \texttt{if}\ (b)\ \texttt{then}\ \{\texttt{cx}(a,c)\}\ \texttt{else}\ \{\texttt{cx}(c,b)\}$, writing $|\phi\rangle$ to indicate the state $^1/\sqrt{2}(|0\rangle + |1\rangle)$ and $|\varphi\rangle$ to indicate $^1/\sqrt{2}(|0\rangle - |1\rangle)$. In this example, we start in node **1** with a state where a and b form a Bell state and are therefore abstracted as being entangled d-inseparable. In nodes **2** and **3**, due to the measurement of b, both a and b collapse to a basis state. Since a and b are d-inseparable, they are both labelled with Z. In node **4**, the concrete state is obtained by merging the semantics of the two paths, while the abstract state is the lub between $CX^\sharp_{a,c}$([({a},0),({b},1),({c},2)],{(0,1):Z,2:X})



(a)

	Concrete state	Abstract state
Start	$\{\|000\rangle_{a,b,c}\}$	[({a},0),({b},1),({c},2)],{(0,1,2):Z})
1	$\{1/\sqrt{2}(\|00\rangle+\|11\rangle)_{a,b}\|\phi\rangle_c\}$	[({a, b},0),({c},1)],{0:X,1:Z})
2	$\{\|11\phi\rangle_{a,b,c}\}$	[({a},0),({b},1),({c},2)],{(0,1):Z,2:X})
3	$\{\|00\phi\rangle_{a,b,c}\}$	[({a},0),({b},1),({c},2)],{(0,1):Z,2:X})
4	$\{\|11\varphi\rangle_{a,b,c}, (\|0\rangle_a\, 1/\sqrt{2}\,\|00\rangle+\|11\rangle)_{a,b,c}\}$	[({a},0),({c},1),({b},1)],{0:Z,1:$\top_{\mathscr{L}}$})

(b)

Figure 5.4: The *prog* CFG (a), and a table representing concrete and abstract states for each node (b), where $|\phi\rangle = 1/\sqrt{2}(|0\rangle + |1\rangle)$ and $|\varphi\rangle = 1/\sqrt{2}(|0\rangle - |1\rangle)$

and $CX^{\sharp}_{c,b}$([({a},0),({b},1),({c},2)],{(0, 1):Z,2:X}), which correspond to:

$$([(\{a\},0),(\{b\},1),(\{c\},2)],\{(0,1):Z,2:X\}) \sqcup ([(\{a\},0),(\{b,c\},1)],\{(0):Z,1:X\}) =$$
$$= [(\{a\},0),(\{c\},1),(\{b\},1)],\{0:Z,1:\top_{\mathscr{L}}\}$$

In both examples, using the labels, we can approximate the variable's state during the execution of the program.

5.6 Discussion

In this chapter, we introduced a static analysis for quantum programming languages, extending the abstract domain for entanglement analysis proposed in [ADM24a]. By enriching the domain with additional labels, our approach offers a more precise and practical method for analysing quantum entanglement. The framework not only detects entangled variables but also provides an abstract representation of the program state, which can serve as a foundation for further analyses. Compared to previous approaches [Per08b], it achieves higher precision while avoiding exponential complexity. Its integration within a while-language setting further demonstrates its versatility and applicability to diverse quantum programming contexts.

Chapter 6

Formal Verification of Variational Quantum Circuits

The computational paradigm of Variational Quantum Circuits (VQCs) provides a general framework for solving a wide range of problems by leveraging the power of quantum computers in synergy with advanced classical optimisation tools [Hav+19; Cer+21; End+21]. At the core of a VQC lies a parametrised quantum circuit with a specific structure, known as the *ansatz* or *variational form*, consisting of a sequence of parametrised gates. During a training procedure, these parameters are iteratively updated based on the classical optimisation of a problem-specific cost function. VQCs enable hybrid quantum-classical algorithms to operate on quantum state data, where quantum circuits are trained analogously to Neural Networks (NNs) to solve complex computational tasks, such as data classification, which we will use in this work to demonstrate our approach to the verification problem. Thanks to their versatility, VQCs have been successfully applied to quantum chemistry (e.g., estimating molecular energies), mathematical problems (e.g., solving systems of equations), combinatorial optimisation, error correction, circuit compilation, and many other domains. They also play a central role in Quantum Machine Learning [SSP15; Bia+17], supporting classification tasks and, more broadly, learning complex patterns from data. Prominent examples include Variational Quantum Support Vector Machines [Hav+19] and Quantum Generative Adversarial Networks [DK18]. However, like NNs, VQCs are susceptible to adversarial inputs, i.e., small perturbations to the input state that lead to unexpected predictions [Sze+13; LDD20; WTD24]. Consequently, understanding and mitigating adversarial vulnerabilities in VQCs is essential to ensure the trustworthy and

reliable deployment of quantum-enhanced learning systems. For example, consider the classification problem with a VQC model that has to classify different road signs, depicted in Figure 6.1.

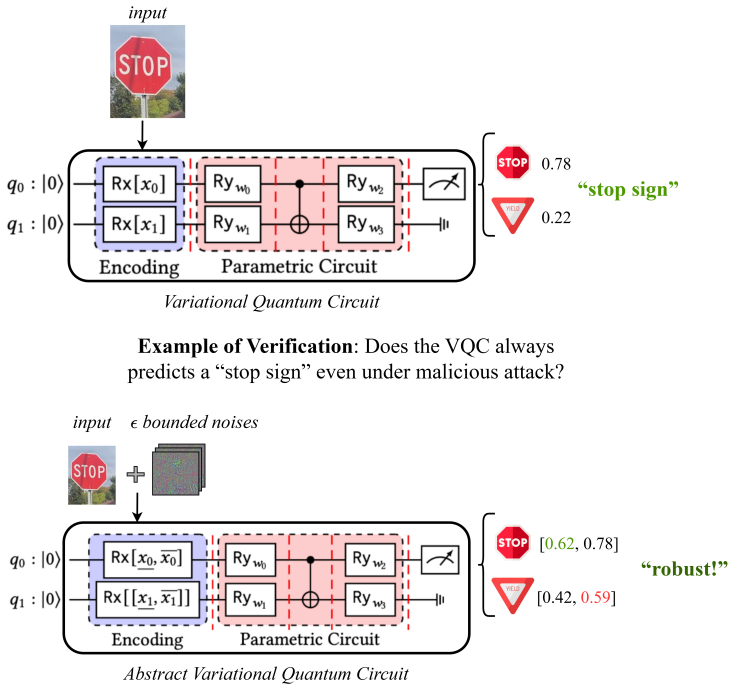


Figure 6.1: Example of robustness verification problem for variational quantum circuit. Top the VQC correctly predicts the "stop sign". On the bottom, a set of ϵ-bounded perturbations is applied to the original input, and the interval-based abstraction presented in this work is applied to evaluate the VQC's robustness in the prediction.

Due to the heuristic and approximate nature of training algorithms, robustness assessments are often purely empirical, which is insufficient for providing formal guarantees.

In this chapter, we present a framework for the theoretical and practical analysis of the formal verification of variational quantum circuits. We build on the analogy between VQC-based supervised learning and classical neural networks by investigating whether techniques developed for the formal verification of classical NNs [Liu+21], particularly those based on abstract interpretation [Sin+19; Geh+18] and interval-based reachability analysis, can be extended to the quantum setting (Figure 6.1 right).

While the *ansatz* of a VQC is defined by a linear learning process based on unitary (and therefore convex) operators, which might suggest that an abstract interpretation approach is simpler than in the classical nonlinear case, this unitary process is applied only after

transforming classical input data into quantum states through an appropriate encoding phase [RD24; SSM21; Hav+19]. This transformation, called quantum embedding or quantum feature map, moves the computation from the classical dataset to the quantum state space, that is, the space of complex vectors with unit norm. To formally verify a VQC, we need a framework capable of transferring an *abstract input*, that is, a set of intervals derived from classical data, into the complex amplitudes characterising quantum states in the circuit. This requires reasoning about abstract states whose amplitudes are intervals of complex numbers. Furthermore, a key challenge in VQCs lies in the *measurement* operation at the end of the circuit, which yields a probability distribution over possible outcomes and is inherently non-invertible: given a distribution, infinitely many quantum states could produce it upon measurement. As in the case of nonlinear neural networks, this non-invertibility prevents backward reasoning on the input set that generated a given output. All these aspects motivate the need for a dedicated formalisation and analysis tailored to VQCs.

6.1 Variational Quantum Circuits

A Variational Quantum Circuit (VQC) consists of three main components: an input encoding (feature map or quantum embedding), a parametric circuit, and a final measurement, followed by classical optimization of a cost function for parameter updates. We describe these components via the small circuit on two qubits depicted in Figure 6.2. The encoding consists of two gates $\texttt{Rx}[x_i]$, $i \in \{0, 1\}$, which are applied to the qubits q_0 and q_1, both in the blank state $|0\rangle$. These gates are rotations around the x-axis by an angle that is proportional to the input features x_i of the data, and, for a generic x, are therefore defined by

$$\texttt{Rx}[x] = \begin{pmatrix} \cos(x/2) & -\iota \sin(x/2) \\ -\iota \sin(x/2) & \cos(x/2) \end{pmatrix}, \tag{6.1}$$

We use the notation $\texttt{Rx}^{q_0}[x_0]$ for the operator $\texttt{I} \otimes \texttt{Rx}[x_0]$, where $\texttt{I}$ is the identity operator, acting as a rotation on q_0 and as the identity on q_1. Starting from the initial state is $|00\rangle = |0\rangle_{q_1} \otimes |0\rangle_{q_0}$, the encoding can be obtained by computing:

$$\begin{aligned} \texttt{Rx}^{q_1}[x_1] \cdot \texttt{Rx}^{q_0}[x_0] \cdot |00\rangle = &\cos(x_1/2)\cos(x_0/2)\,|00\rangle - \iota\cos(x_1/2)\sin(x_0/2)\,|01\rangle + \\ &- \iota\sin(x_1/2)\cos(x_0/2)\,|10\rangle - \sin(x_1/2)\sin(x_0/2)\,|11\rangle\,. \end{aligned}$$

This shows how the encoding gates map the values of x_i in the amplitude of a quantum state, on which we can now operate with the parametric quantum circuit. As a concrete instantiation, consider some classical data represented by the vector $[x_0, x_1]^T = [6.0, 2.7]^T$.

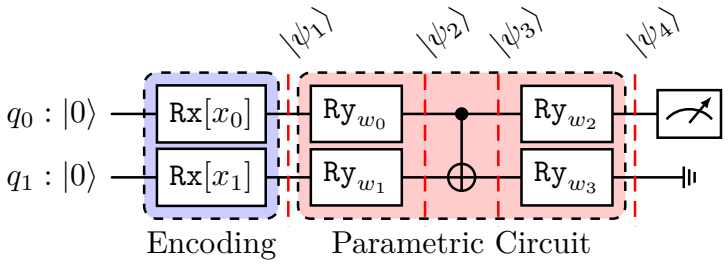


Figure 6.2: A Variational Quantum Circuit. The input x_0, x_1 are encoded in a quantum state by $\texttt{Rx}[x_0]$ and $\texttt{Rx}[x_1]$. The weights w_i are the trainable parameters optimized during the training. The final measurement provides the values required for classical optimization.

The encoding phase applies $\texttt{Rx}[6.0]$ to q_0 and $\texttt{Rx}[2.7]$ to q_1, with $|q_0 q_1\rangle = |00\rangle$, obtaining:

$$|\psi_1\rangle = (\texttt{Rx}^{q_1}[2.7] \cdot \texttt{Rx}^{q_0}[6.0])\,|00\rangle = -0.22\,|00\rangle - 0.03\iota\,|01\rangle + 0.97\iota\,|10\rangle - 0.14\,|11\rangle\,. \tag{6.2}$$

On this quantum state, we can now apply the parametric quantum circuit, which, in our example, is represented by the *ansatz* in Figure 6.2 with trainable parameters $w_0, \ldots, w_3$. As with Neural Networks, during the training phase, the parameters w_i are optimized to minimize the classification error. To distinguish the rotation gates used in the input encoding phase from those in the variational part, we will denote the latter by $\texttt{R}_w$, where w is the weight optimized during training. When verifying the circuit, this is a fixed constant, as verification is performed after the training phase. In our example, the training phase produces $w_0 = 0.99$, $w_1 = -0.50$, $w_2 = 3.27$ and $w_3 = -0.69$. Thus, by first applying to $|\psi_1\rangle$ the gates $\texttt{Ry}^{q_0}_{0.99}$, $\texttt{Ry}^{q_1}_{-0.50}$*, and then $\texttt{CX}^{q_0,\,q_1}$ followed by $\texttt{Ry}^{q_0}_{3.27}$ and $\texttt{Ry}^{q_1}_{-0.69}$, we obtain the final state $|\psi_4\rangle$:

$$|\psi_4\rangle = (0.14 - 0.49\iota)\,|00\rangle - (0.11 - 0.46\iota)\,|01\rangle + (0.08 + 0.03\iota)\,|10\rangle + (0.17 + 0.70\iota)\,|11\rangle\,. \tag{6.3}$$

The last step is the measurement of the qubits in the computational basis. We have four possible outcomes: 00, 01, 10, and 11. By computing the squared modulus† of the amplitude of each state, we obtain the distribution $(0.26, 0.21, 0.01, 0.52)$ over the possible outcomes $\{00, 01, 10, 11\}$. Based on this, a classification can be made by looking at the result of q_0. Since q_0 is the least significant qubit, the probability of measuring q_0 in state 1 corresponds to measuring 01 or 11 and can be computed as $0.21 + 0.52 = 0.73$. Similarly, the probability of measuring it in state 0 (i.e., measuring 00 or 10) is $0.26 + 0.01 = 0.27$. This means that

* We recall that $\texttt{Ry}_\theta = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}$

† Given a complex number $z = a + ib$, the squared modulus $|z|^2 = a^2 + b^2$.

we can classify the input $[x_0, x_1]^T = [6.0, 2.7]^T$ as class represented by the result 1 with a probability of 0.73.

In the training of a VQC, the measurement outcome of the qubit is typically used to compute a cost function, which is then optimized by using classical optimization tools to update the parameters of the circuits. For the verification task, which is the purpose of this chapter, we can concentrate on trained VQC, where the classical optimization phase has already taken place.

6.2 A language for Variational Quantum Circuits

In order to formally verify a VQC, we introduce a language, which we call $\mathscr{L}_{\texttt{VQC}}$, defining the structure and the behavior of VQCs. For this language, we first define a syntax (establishing the structure of a VQC), and then we give a semantics, inductively defined on the constructs of the syntax, by using a function-based representation of quantum states.

6.2.1 Syntax

Let q, q_i, q_j denote the VQC qubits and x a classical input variable. We define $\mathscr{L}_{\texttt{VQC}}$ as the language generated by the grammar:

$$
\begin{aligned}
&\texttt{E} := \texttt{Rx}^q[x] \mid \texttt{Ry}^q[x] \mid \texttt{Rz}^q[x] && \text{(Encoding operations)}\\
&\texttt{U} := \texttt{Rx}^q_w \mid \texttt{Ry}^q_w \mid \texttt{Rz}^q_w \mid \texttt{CX}^{q_i,q_j} && \text{(Parametric operations)}\\
&\texttt{s} := \texttt{E} \mid \texttt{U} \mid \texttt{s};\texttt{s} && \text{(Quantum statements)}\\
&\texttt{c} := \texttt{s};\texttt{M} && \text{(VQC)}
\end{aligned}
$$

A VQC $\texttt{c}$ is any element in the language $\mathscr{L}_{\texttt{VQC}}$ consisting of a quantum circuit $\texttt{s}$, composed by quantum statements, followed by a final measurement operation $\texttt{M}$. A quantum statement is a sequential composition of parametric and encoding operations. $\texttt{E}$ represents the encoding operations, used to embed input values into quantum states. Each operation $\texttt{E}$ represents a rotation around one of the three axes of the Bloch sphere, with an angle determined by the input variable x and applied to qubit q. $\texttt{U}$ defines the operations used in the parametric (trainable) part of the circuit. It includes single-qubit rotations and the controlled-NOT operation $\texttt{CX}^{q_i,q_j}$, where q_i is the control and q_j the target qubit, with $q_i \neq q_j$. We syntactically distinguish between $\texttt{U}$ and $\texttt{E}$ to reflect their different roles within the circuit: Encoding operations depend on input variables, and are used to embed input data into quantum states; Parametric operations are parameterized by constants w, representing the trainable weights optimized during the training phase. For instance, the VQC in Figure 6.2

can be written as:

$$\texttt{Rx}^{q_0}[x_0]; \texttt{Rx}^{q_1}[x_1]; \texttt{Ry}^{q_0}_{w_0}; \texttt{Ry}^{q_1}_{w_1}; \texttt{CX}^{q_0,q_1}; \texttt{Ry}^{q_0}_{w_2}; \texttt{Ry}^{q_1}_{w_3}; \texttt{M}$$

We recall that, even if we are considering only the control-not and one-qubit rotation gates, we can represent in $\mathscr{L}_{\texttt{vqc}}$ any possible quantum circuit [Bar+95].

6.2.2 Semantics

To define a semantics for $\mathscr{L}_{\texttt{vqc}}$, we first need to specify the *domain* of the denotations of data and then a function modeling how the computation transforms such data.

Semantic Data Domains

Let $\texttt{QV}_\texttt{c}$ represent the set of qubits in a VQC $\texttt{c} \in \mathscr{L}_{\texttt{vqc}}$. Each qubit $q \in \texttt{QV}_\texttt{c}$ is associated with a two-dimensional Hilbert space $\mathscr{H}_q$. Consequently, the global vector space of the program is denoted as $\mathscr{H}_{\texttt{QV}_\texttt{c}} = \bigotimes_{q \in \texttt{QV}_\texttt{c}} \mathscr{H}_q$. The program state is then represented as a vector $|\psi\rangle \in \mathscr{H}_{\texttt{QV}_\texttt{c}}$. In this work, we use an alternative representation rather than the standard vector formalism, which is more suitable for defining the abstract domain. Since a quantum state corresponds to a superposition of basis states, each one associated with a complex amplitude, we represent quantum states ψ as mappings from the set $\mathscr{B}_\texttt{c}$ of standard basis vectors of $\mathscr{H}_{\texttt{QV}_\texttt{c}}$, to complex amplitudes in $\mathbb{C}$. Formally, a state $\psi \in \Psi_\texttt{c} \stackrel{\text{def}}{=} \mathscr{B}_\texttt{c} \to \mathbb{C}$ can be represented as $\psi = \lambda e \in \mathscr{B}_\texttt{c}.\ z_e$ ($z_e \in \mathbb{C}$), where the normalization condition $\sum_{e \in \mathscr{B}_\texttt{c}} |\psi(e)|^2 = 1$ must hold.

This functional representation is equivalent to the standard vector representation. Throughout the chapter, we use ψ to denote the state as a mapping and $|\psi\rangle$ to refer to its equivalent vector representation. For instance, consider the state $|\psi_1\rangle$ defined in Equation 6.2. Since this is a two-qubit state, the basis set in mapping notation is $\mathscr{B}_\texttt{c} = \{00, 01, 10, 11\}$. Then, the corresponding mapping is $\psi_1 = \{00 \mapsto -0.22,\ 01 \mapsto -0.03\iota,\ 10 \mapsto +0.97\iota,\ 11 \mapsto -0.14\}$.

States are not the only data on which a VQC works. Indeed, we must also consider the input for the variable x. In particular, we need to assign them real values—that is, we require an *input environment*. Given $\texttt{c} \in \mathscr{L}_{\texttt{vqc}}$, let $\texttt{CV}_\texttt{c}$ denote the set of classical input variables. We define an input environment as a mapping $\sigma \in \Sigma_\texttt{c} \stackrel{\text{def}}{=} \texttt{CV}_\texttt{c} \to \mathbb{R}$, associating each variable in $\texttt{CV}_\texttt{c}$ with a real-valued input in $\mathbb{R}$. For instance, in the example described in Section 6.1, the input environment is given by: $\sigma = \{x_0 \mapsto 6.0,\ x_1 \mapsto 2.7\}$.

Quantum Statements Semantics

We can define the semantics of a quantum statement $\texttt{s}$ as a function $[\![\texttt{s}]\!] : \Sigma_\texttt{c} \to (\Psi_\texttt{c} \to \Psi_\texttt{c})$, which, given an input environment, transforms an initial state $\psi \in \Psi_\texttt{c}$ into a resulting

quantum state in $\Psi_{\mathtt{c}}$.

Parametric Operations $\mathtt{U}$**.** The semantics of every parametric operation corresponds to a matrix that describes the effect of this operation (whether a single-qubit rotation or a controlled-Not) on the full Hilbert space of the program. For example, consider the parametric operation $\mathtt{Ry}^{q}_{w}$. $\mathtt{Ry}_{w}$ corresponds to a 2×2 unitary matrix acting on a single qubit. However, to represent the semantics of the rotation in the full program space, we must define a unitary matrix of size $2^{|\mathtt{QV_c}|} \times 2^{|\mathtt{QV_c}|}$ for $\mathtt{Ry}^{q}_{w}$. This larger matrix models the action of $\mathtt{Ry}_{w}$ on qubit q while acting as the identity on all other qubits in $\mathtt{QV_c} \setminus \{q\}$. Concretely, this is achieved by taking the tensor product of the 2×2 matrix representing $\mathtt{Ry}_{w}$ with identity matrices corresponding to the remaining qubits.

Given an operation in $\mathtt{U}$, we abuse notation considering as $\mathtt{U} \in \mathbb{C}^{2^{|\mathtt{QV_c}|} \times 2^{|\mathtt{QV_c}|}}$ the matrix that describes the effect of the operation in the full Hilbert space of the program. Given this matrix, each entry $\mathtt{U}_{e_1,\, e_2} \in \mathbb{C}$ denotes the amplitude of transitioning from basis state e_2 to e_1[‡], and applying $\mathtt{U}$ to a quantum state $|\psi\rangle \in \mathbb{C}^{|\mathscr{B}_{\mathtt{c}}|}$ corresponds to the matrix-vector product $\mathtt{U} \cdot |\psi\rangle$. To define the semantics of this transformation in our setting, we must express matrix multiplication using the mapping-based formalism employed to represent abstract states. Given an operation in $\mathtt{U}$, its semantics is defined as a function $[\![\mathtt{U}]\!] : \mathbb{\Sigma}_{\mathtt{c}} \to (\Psi_{\mathtt{c}} \to \Psi_{\mathtt{c}})$, defined as:

$$[\![\mathtt{U}]\!]_{\sigma} \stackrel{\text{def}}{=} \lambda\psi. \left(\lambda e \in \mathscr{B}_{\mathtt{c}}. \sum_{e' \in \mathscr{B}_{\mathtt{c}}} \mathtt{U}_{e,\, e'} \cdot \psi(e') \right) \tag{6.4}$$

Since $\mathtt{U}$ is not an encoding operation, its semantics is independent of the input environment σ. For instance, consider a one qubit quantum state $|\psi\rangle_{q} = [a, b]^{T}$ (equivalently represented as $\psi_q = \{0 \mapsto a, 1 \mapsto b\}$), and the operation described by the matrix:

$$\mathtt{G} = \begin{bmatrix} u_{0,0} & u_{0,1} \\ u_{1,0} & u_{1,1} \end{bmatrix}.$$

In vector formalism the application of $\mathtt{G}$ to $|\psi\rangle$ is given by:

$$\mathtt{G} \cdot |\psi\rangle = \begin{bmatrix} u_{0,0}a + u_{0,1}b \\ u_{1,0}a + u_{1,1}b \end{bmatrix},$$

that correspond, in the mapping notation, to:

$$[\![\mathtt{G}]\!](\psi) = \left\{ \begin{matrix} 0 \mapsto u_{0,0}a + u_{0,1}b, \\ 1 \mapsto u_{1,0}a + u_{1,1}b \end{matrix} \right\} = \left\{ \begin{matrix} 0 \mapsto u_{0,0} \cdot \psi(0) + u_{0,1} \cdot \psi(1), \\ 1 \mapsto u_{1,0} \cdot \psi(0) + u_{1,1} \cdot \psi(1) \end{matrix} \right\}$$

‡ We recall that the bit strings e_1 and e_2 that represent standard basis elements can be used as indices to the rows and columns of the matrix $\mathtt{U}$, respectively.

Encoding Operations $\mathtt{E}$. We can adopt an approach similar to the one used for parametric operations. $\mathtt{E}$ represents single-qubit rotations by an angle depending on the value of x. This means that every rotation is described by a matrix over functions, $\mathtt{Fun}[x]^{2\times 2}$, where $\mathtt{Fun}[x] = \{f[x] \mid f : \mathbb{R} \to \mathbb{C},\ x \in \mathtt{CV_c}\}$ is a set of complex-valued functions over the variable x. The matrix $\mathtt{Rx}[x]$ in Equation 6.1 is an example of such a rotation in $\mathtt{E}$. Abusing notation, by $\mathtt{E}$ we refer to the matrix modeling the encoding operation acting on the entire system: the matrix of the specific rotation on a qubit in tensor with the identity on the rest. Accordingly, $\mathtt{E} \in \mathtt{Fun}[x]^{2^{|\mathtt{QV_c}|}\times 2^{|\mathtt{QV_c}|}}$.

In order to define the semantics of $\mathtt{E}$, we first need to evaluate the functions $\mathtt{f}[x] \in \mathtt{Fun}[x]$ with respect to an input environment $\sigma \in \mathbb{\Sigma}_\mathtt{c}$. We define the interpretation function $(\!|\mathtt{f}[x]|\!) : \mathbb{\Sigma}_\mathtt{c} \to \mathbb{C}$ as: $(\!|\mathtt{f}[x]|\!)_\sigma = \mathtt{f}(\sigma(x)) \in \mathbb{C}$, that is, we evaluate the function $\mathtt{f}$ on the input value $\sigma(x)$. Given an encoding operation $\mathtt{E}$, each matrix entry $\mathtt{E}_{e_1,\,e_2}$ is a function in $\mathtt{Fun}[x]$, mapping an input value to a complex coefficient. The semantics of $\mathtt{E}$ is a function $[\![\mathtt{E}]\!] : \mathbb{\Sigma}_\mathtt{c} \to (\mathbb{\Psi}_\mathtt{c} \to \mathbb{\Psi}_\mathtt{c})$ defined as:

$$[\![\mathtt{E}]\!]_\sigma \stackrel{\text{def}}{=} \lambda\psi.\ (\lambda e \in \mathscr{B}_\mathtt{c}.\ \sum_{e' \in \mathscr{B}_\mathtt{c}} (\!|\mathtt{E}_{e,\,e'}|\!)_\sigma \cdot \psi(e')) \tag{6.5}$$

Sequential Composition. The last operator for building a VQC is the sequential composition of statements $\mathtt{s}_1; \mathtt{s}_2$. In this case, the semantics is trivially the functional composition of the semantics:

$$[\![\mathtt{s}_1;\ \mathtt{s}_2]\!]_\sigma \stackrel{\text{def}}{=} [\![\mathtt{s}_2]\!]_\sigma \circ [\![\mathtt{s}_1]\!]_\sigma$$

The Semantics of a VQC

Since a VQC $\mathtt{c} \in \mathscr{L}_{\mathtt{VQC}}$ is built as $\mathtt{c} := \mathtt{s}; \mathtt{M}$, its semantics requires a denotation for the final measurement operator. This can be defined as a function that, given a quantum state, returns a probability distribution over the set of possible results. In our setting, we define $\mathbb{\Delta}_\mathtt{c} \stackrel{\text{def}}{=} \mathscr{B}_\mathtt{c} \to \mathbb{R}$ (with metavariable ρ), the set of all possible probability distributions over the bases $\mathscr{B}_\mathtt{c}$. The measurement semantics can be expressed (abusing notation) as a function $[\![\mathtt{M}]\!] : \mathbb{\Psi}_\mathtt{c} \to \mathbb{\Delta}_\mathtt{c}$, formally defined as:

$$[\![\mathtt{M}]\!] \stackrel{\text{def}}{=} \lambda\psi.\ (\lambda e \in \mathscr{B}_\mathtt{c}.\ |\psi(e)|^2). \tag{6.6}$$

For instance, let us consider the state in Equation 6.3, that in the mapping representation is $\psi_4 = \{00 \mapsto (0.14 - 0.49\iota), 01 \mapsto -(0.11 - 0.46\iota), 10 \mapsto (0.08 + 0.03\iota), 11 \mapsto (0.17 + 0.70\iota)\}$. The result of the application measurement operators is:

$$[\![\mathtt{M}]\!](\psi_4) = \{00 \mapsto 0.26, 01 \mapsto 0.21, 10 \mapsto 0.01, 11 \mapsto 0.52\} \in \mathbb{\Delta}_{q_0,\,q_1}$$

Finally, the concrete semantics of the entire $\texttt{c} = \texttt{s};\texttt{M}$, is the composition of the semantics of the measurement applied to the result of operator $\texttt{s}$ semantics, obtaining (again abusing notation) a function $[\![\texttt{c}]\!]\colon \mathbb{\Sigma}_{\texttt{c}} \to (\mathbb{\Psi}_{\texttt{c}} \to \mathbb{\Delta}_{\texttt{c}})$ defined as:

$$[\![\texttt{c}]\!]_\sigma \stackrel{\text{def}}{=} \lambda\psi.\, [\![\texttt{M}]\!] \circ [\![\texttt{s}]\!]_\sigma(\psi).$$

When verifying a VQC, we execute the circuit on a set of input values, which corresponds to a specific input and all possible values obtained by perturbing it. In the concrete domain, this would involve executing the circuit for every possible input. To formalize this, we define the concrete semantics of the circuit in a collecting style, i.e., as a function $[\![\texttt{c}]\!] : \wp(\mathbb{\Sigma}_{\texttt{c}}) \to (\wp(\mathbb{\Psi}_{\texttt{c}}) \to \wp(\mathbb{\Delta}_{\texttt{c}}))$. We abuse notation by using the same semantic notation for the additive lift of $[\![\cdot]\!]$ to sets of environments and states, i.e., $[\![\texttt{c}]\!]_\Sigma(\Psi) \stackrel{\text{def}}{=} \{\, [\![\texttt{c}]\!]_\sigma(\psi) \mid \sigma \in \Sigma, \psi \in \Psi \,\}$, where $\Sigma \in \wp(\mathbb{\Sigma}_{\texttt{c}})$ and $\Psi \in \wp(\mathbb{\Psi}_{\texttt{c}})$ represent a set of initial states of the VQC.

6.3 Abstracting $\mathscr{L}_{\texttt{vqc}}$ Semantics

In many fields of computation, the concrete semantics is usually an intractable model of computation due to computability or computational complexity reasons. In this section, we follow the idea widely exploited both in programming languages and in NN [CC79; Geh+18], consisting of executing the computational system on *approximated data*, e.g., intervals of numerical values. In order to approximate the computation, we first need to abstract the data. We observe that inputs (environments) and outputs (distributions) are mappings to real values, while the VQC works on states, which are mappings to complex values. This means that we have to split the abstraction process, namely, we have to introduce the real intervals abstraction for environments and distributions, and the complex interval abstraction for the states.

6.3.1 Abstracting Environments (inputs) and Distributions (outputs)

Let us first extend the standard abstract domain of intervals to real values, with a slight change consisting of considering only bounded intervals.

Reals Interval Domain

Let us consider the domain of closed [Cou21a] and bounded [PT10] intervals on $\mathbb{R}$, i.e., $\mathbb{RI} \stackrel{\text{def}}{=} \{\, [r_l, r_u] \mid r_l, r_u \in \mathbb{R} \,\} \cup \{\varnothing, \mathbb{R}\}$. As it happens for integer intervals, also this domain can be characterized as an abstraction of the powerset of reals $\langle \wp(\mathbb{R}), \subseteq \rangle$ [Cou21a]. We

define two functions $\alpha_{\texttt{RI}} : \wp(\mathbb{R}) \to \mathbb{RI}$ and $\gamma_{\texttt{RI}} : \mathbb{RI} \to \wp(\mathbb{R})$ as: Let $I \in \mathbb{RI}$

$$\gamma_{\texttt{RI}}(I) \stackrel{\text{def}}{=} \begin{cases} \{r \mid r_l \leq r \leq r_u\} & \text{if } I = [r_l, r_u] \\ I & \text{if } I = \varnothing \text{ or } I = \mathbb{R} \end{cases} \qquad \alpha_{\texttt{RI}}(X) \stackrel{\text{def}}{=} \bigcap \{\, I \in \mathbb{RI} \mid X \subseteq \gamma_{\texttt{RI}}(I) \,\}$$

Note that $\alpha_{\texttt{RI}}$ is well-defined since closed intervals on $\mathbb{R}$ are closed sets in the corresponding topological space, and the intersection of any family of closed sets is a closed set, with $\varnothing$ and $\mathbb{R}$ closed sets. In general, let $X \in \wp(\mathbb{R})$, $\alpha_{\texttt{RI}}(X) = [\min(X), \max(X)]$ if both $\min(X)$ and $\max(X)$ exist in X; If at least one of them does not exist since X has an *open* bound, e.g., $X = \{\, r \mid 2 < r \leq 10 \,\}$, then the abstraction closes it, namely $\alpha_{\texttt{RI}}(X) = [2, 10]$; If, at least one bound does not exist, e.g., $X = \{\, r \in \mathbb{R} \mid r \geq 10 \,\}$, then $\alpha_{\texttt{RI}}(X) = \mathbb{R}$; If $X = \varnothing$ then the abstraction is $\varnothing$. When $\alpha_{\texttt{RI}}(X) = [l, u]$ we abuse notation writing $l = \min(X)$ and $u = \max(X)$ [PT10].

The following results come from the fact that $\gamma_{\texttt{RI}}$ is a co-additive function over $\mathbb{RI}$, and therefore, by construction, $\alpha_{\texttt{RI}}$ is its adjoint function forming a Galois Insertion.

Proposition 6.1. $\langle \wp(\mathbb{R}), \subseteq \rangle \underset{\alpha_{\texttt{RI}}}{\overset{\gamma_{\texttt{RI}}}{\leftrightarrows}} \langle \mathbb{RI}, \leq_{\mathbb{RI}} \rangle$, *where* $\leq_{\mathbb{RI}}$ *is standard inclusion between intervals.*

In [Cou21a; MKC09] we can find the whole arithmetic on real intervals. Here we will use only multiplication (extended to matrices) and sum, which we denote, abusing notation, as $\cdot$ and $+$.

Abstract Environments

Environments map input variables to real values. In order to abstract the VQC computation to intervals, we have to define *abstract environments* $\overline{\mathtt{S}}$ mapping the input variables of $\mathtt{c} \in \mathscr{L}_{\texttt{VQC}}$ (i.e., those in $\mathtt{CV_c}$) to the *intervals* of values in $\mathbb{RI}$ representing the range of possible values for that variable. This abstraction is performed on the collection of reals associated with each variable (due to a set of possible input environments). For example, if a variable $x \in \mathtt{CV_c}$ can take values between 0 and $r \in \mathbb{R}$, the abstract environment would map x to the interval $[0, r]$. Formally, $\overline{\mathtt{S}} \in \overline{\mathbb{S}}_{\mathtt{c}} \stackrel{\text{def}}{=} \mathtt{CV_c} \to \mathbb{RI}$ can be defined by abstract interpretation on the power domain of environments. Let $\alpha_{\texttt{env}} : \wp(\mathbb{\Sigma}_{\mathtt{c}}) \to \overline{\mathbb{S}}_{\mathtt{c}}$ defined as follows:

$$\alpha_{\texttt{env}}(\Sigma) \stackrel{\text{def}}{=} \lambda x \in \mathtt{CV_c}.\, \alpha_{\texttt{RI}}\big(\{\, \sigma(x) \mid \sigma \in \Sigma \,\}\big),$$

This function abstracts a set of inputs (i.e., a set of static environments) into a single abstract environment $\overline{\mathtt{S}}$. The concretization is the standard adjoint $\gamma_{\texttt{env}} : \overline{\mathbb{S}}_{\mathtt{c}} \to \wp(\mathbb{\Sigma}_{\mathtt{c}})$, namely $\forall \overline{\mathtt{S}} \in \overline{\mathbb{S}}_{\mathtt{c}}$ we have $\gamma_{\texttt{env}}(\overline{\mathtt{S}}) \stackrel{\text{def}}{=} \{\, \sigma \in \mathbb{\Sigma}_{\mathtt{c}} \mid \forall x \in \mathtt{CV_c}.\, \sigma(x) \in \gamma_{\texttt{RI}}(\overline{\mathtt{S}}(x)) \,\}$.

Proposition 6.2. $\langle \wp(\mathbb{\Sigma}_c), \subseteq \rangle \underset{\alpha_{\text{env}}}{\overset{\gamma_{\text{env}}}{\leftrightarrows}} \langle \overline{\mathbb{S}}_c, \dot{\leq}_{\mathbb{RI}} \rangle$, *where* $\forall \overline{\mathbb{S}}_1, \overline{\mathbb{S}}_2 \in \overline{\mathbb{S}}_c$ *we have* $\overline{\mathbb{S}}_1 \dot{\leq}_{\mathbb{RI}} \overline{\mathbb{S}}_2$ *iff* $\forall x \in \texttt{CV}_c.\ \overline{\mathbb{S}}_1(x) \leq_{\mathbb{RI}} \overline{\mathbb{S}}_2(x)$.

Abstract Distributions

The output of the VQC semantics is a probability distribution, which must also be abstracted in order to return intervals of probabilities when measuring an abstract state. Formally, let us define $\overline{\mathbb{D}}_{\texttt{c}} \stackrel{\text{def}}{=} \mathscr{B}_{\texttt{c}} \to \mathbb{RI}$, the abstract domain of probability distributions on intervals, pointwise ordered by $\dot{\leq}_{\mathbb{RI}}$. We can define the abstraction function $\alpha_{\texttt{dist}} : \wp(\mathbb{\Delta}_{\texttt{c}}) \to \overline{\mathbb{D}}_{\texttt{c}}$, between sets of probability distributions $\langle \wp(\mathbb{\Delta}_{\texttt{c}}), \subseteq \rangle$ and abstract interval distributions $\langle \overline{\mathbb{D}}_{\texttt{c}}, \dot{\leq}_{\mathbb{RI}} \rangle$. Given $\Delta \in \wp(\mathbb{\Delta}_{\texttt{c}})$, we define:

$$\alpha_{\texttt{dist}}(\Delta) \stackrel{\text{def}}{=} \lambda e \in \mathscr{B}_{\texttt{c}}.\ \alpha_{\texttt{RI}}\left(\{\rho(e) \mid \rho \in \Delta\}\right).$$

The concretization function $\gamma_{\texttt{dist}} : \overline{\mathbb{D}}_{\texttt{c}} \to \wp(\mathbb{\Delta}_{\texttt{c}})$ is the standard adjoint, i.e., $\forall \varrho^\sharp \in \overline{\mathbb{D}}_{\texttt{c}}.\gamma_{\texttt{dist}}(\varrho^\sharp) \stackrel{\text{def}}{=} \{\ \rho \in \mathbb{\Delta}_{\texttt{c}} \mid \forall e \in \mathscr{B}_{\texttt{c}}.\ \rho(e) \in \gamma_{\texttt{RI}}(\varrho^\sharp(e))\ \}$.

For instance, the mapping $\varrho^\sharp = \{00 \mapsto [0.126, 0.50], 01 \mapsto [0.116, 0.460], 10 \mapsto [0, 0.140], 11 \mapsto [0.291, 0.761]\}$ in Equation 6.8, express the abstract state where we can obtain 00 with probability (w.p.) $[0.126, 0.50]$, 01 w.p. $[0.116, 0.460]$, 10 w.p. $[0, 0.140]$, and 11 w.p. $[0.291, 0.761]$.

6.3.2 Abstracting Quantum States

In order to abstract states, mapping quantum variables to complex values, into the interval domain, we need to extend the abstract domain of intervals to complex values, i.e., to pairs of real intervals.

Complex Interval Domain

Let us consider the set of complex numbers $\mathbb{C}$. Given a generic complex number $z = r_1 + \imath r_2 \in \mathbb{C}$, we denote by $\text{Re}(z) = r_1$ its real part, and by $\text{Im}(z) = r_2$ its imaginary part. Analogously, given a $Z \in \wp(\mathbb{C})$, $\text{Re}(Z) \stackrel{\text{def}}{=} \{\ \text{Re}(z) \mid z \in Z\ \}$ and $\text{Im}(Z) \stackrel{\text{def}}{=} \{\ \text{Im}(z) \mid z \in Z\ \}$. Now, we can abstract complex numbers to intervals simply by keeping the information that z is determined by the pairs of real values r_1 and r_2. Following this idea, we define the abstract domain of complex intervals as $\mathbb{CI} \stackrel{\text{def}}{=} \mathbb{RI} \times \mathbb{RI}$. Hence, we can trivially extend the interval abstraction and concretization functions to complex numbers as: Let $Z \in \wp(\mathbb{C})$ and

$R, I \in \mathbb{RI}$

$$\alpha_{\texttt{CI}} : \wp(\mathbb{C}) \to \mathbb{CI},\ \alpha_{\texttt{CI}}(Z) \stackrel{\text{def}}{=} \langle \alpha_{\texttt{RI}}(\text{Re}(Z)), \alpha_{\texttt{RI}}(\text{Im}(Z)) \rangle$$
$$\gamma_{\texttt{CI}} : \mathbb{CI} \to \wp(\mathbb{C}),\ \gamma_{\texttt{CI}}(R, I) \stackrel{\text{def}}{=} \{\ z \in \mathbb{C} \mid \text{Re}(z) \in \gamma_{\texttt{RI}}(R),\ \text{Im}(z) \in \gamma_{\texttt{RI}}(I)\ \}$$

Proposition 6.3. $\langle \wp(\mathbb{C}), \subseteq \rangle \underset{\alpha_{\texttt{CI}}}{\overset{\gamma_{\texttt{CI}}}{\leftrightarrows}} \langle \mathbb{CI}, \leq_{\mathbb{CI}} \rangle$, *where* $\forall \langle R_1, I_1 \rangle, \langle R_2, I_2 \rangle \in \mathbb{CI}$ *we have* $\langle R_1, I_1 \rangle \leq_{\mathbb{CI}} \langle R_2, I_2 \rangle$ *iff* $R_1 \leq_{\mathbb{RI}} R_2$ *and* $I_1 \leq_{\mathbb{RI}} I_2$.

We abuse notation by using the same operator introduced on real intervals also on complex intervals, with the only difference of being point wise applied to the elements of the pair, i.e., $\langle [a_1, b_1], [c_1, d_1] \rangle + \langle [a_2, b_2], [c_2, d_2] \rangle = \langle [a_1, b_1] + [a_2, b_2], [c_1, d_1] + [c_2, d_2] \rangle$, analogously for $\cdot$ [MKC09; GH71].

Abstract States

Similarly to what we did for the abstract environments, we now need to map each standard basis vector to an interval of complex values in order to abstract states. We define the set of abstract states $\overline{\mathbb{V}}_{\texttt{c}} = \mathscr{B}_{\texttt{c}} \to \mathbb{CI}$ mapping each standard basis $e \in \mathscr{B}_{\texttt{c}}$ to a complex abstract interval $\langle [a, a'], [b, b'] \rangle \in \mathbb{CI}$. This domain $\overline{\mathbb{V}}_{\texttt{c}}$ can be obtained as the abstraction function $\alpha_{\texttt{st}} : \wp(\Psi_{\texttt{c}}) \to \overline{\mathbb{V}}_{\texttt{c}}$: Let $\Psi \in \wp(\Psi_{\texttt{c}})$

$$\alpha_{\texttt{st}}(\Psi) \stackrel{\text{def}}{=} \lambda e \in \mathscr{B}_{\texttt{c}}.\ \alpha_{\texttt{CI}}(\{\ \psi(e) \mid \psi \in \Psi\ \})$$

The concretization function $\gamma_{\texttt{st}} : \overline{\mathbb{V}}_{\texttt{c}} \to \wp(\Psi_{\texttt{c}})$ is the standard adjoint, i.e., defined as $\forall\, \overline{\texttt{V}} \in \overline{\mathbb{V}}_{\texttt{c}}.\ \gamma_{\texttt{st}}(\overline{\texttt{V}}) \stackrel{\text{def}}{=} \{\ \psi \in \Psi_{\texttt{c}} \mid \forall e \in \mathscr{B}_{\texttt{c}}.\psi(e) \in \gamma_{\texttt{CI}}(\overline{\texttt{V}}(e))\ \}$.

Proposition 6.4. $\langle \wp(\Psi_c), \subseteq \rangle \underset{\alpha_{\texttt{st}}}{\overset{\gamma_{\texttt{st}}}{\leftrightarrows}} \langle \overline{\mathbb{V}}_c, \dot{\leq}_{\mathbb{CI}} \rangle$ *(*$\dot{\leq}_{\mathbb{CI}}$ *is the point wise ordering induced by* $\leq_{\mathbb{CI}}$*).*

An example of an abstract state is in Equation 6.7, where,

$$\overline{\mathbb{V}}_4(00) = \langle [-0.159, 0.433], [-0.559, -0.355] \rangle,$$

meaning that the real part of the amplitude associated with 00 ranges between -0.159 and 0.433 while the imaginary part ranges between -0.559 and -0.355.

6.3.3 Abstract Semantics of $\mathscr{L}_{\texttt{VQC}}$

Starting from an abstract environment $\overline{\texttt{S}}$ and an abstract state $\overline{\texttt{V}}$, both working on intervals, we can define the abstract semantics of a VQC $\texttt{c} \in \mathscr{L}_{\texttt{VQC}}$ executing $\texttt{c}$ on intervals.

We first have to define an abstract evaluation of functions $\texttt{f}[x] \in \texttt{Fun}[x]$ w.r.t. the abstract

environment, and therefore we have to compute $\texttt{f}$ on intervals. Let $\overline{\mathtt{S}} \in \overline{\mathbb{S}}_{\mathtt{c}}$, we define $(\!|\texttt{f}[x]|\!)^\sharp : \overline{\mathbb{S}}_{\mathtt{c}} \to \mathbb{CI}$ as an approximation of $\texttt{f}$ on intervals: $(\!|\texttt{f}[x]|\!)^\sharp_{\overline{\mathtt{S}}} \stackrel{\text{def}}{=} \alpha_{\texttt{CI}} \circ \texttt{f} \circ \gamma_{\texttt{CI}}(\overline{\mathtt{S}}(x)) \in \mathbb{CI}$. This is the best correct approximation of (the additive lift to sets of) $\texttt{f}$ on $\mathbb{CI}$.

Given $\texttt{c} = \texttt{s}; \texttt{M} \in \mathscr{L}_{\texttt{vqc}}$, in order to define its abstract semantics, we need to define separately the abstract semantics of quantum statements $\texttt{s}$ and of the measurement operator $\texttt{M}$.

Definition 6.1 (Abstract Quantum Statements Semantics)**.** *For each statement s, the abstract semantics is the function $[\![s]\!]^\sharp : \overline{\mathbb{S}}_c \to (\overline{\mathbb{V}}_c \to \overline{\mathbb{V}}_c)$ is defined as follows: Let $\overline{\mathtt{V}} \in \overline{\mathbb{V}}_c$*

(1) $[\![\texttt{U}]\!]^\sharp_{\overline{\mathtt{S}}} \stackrel{def}{=} \lambda\overline{\mathtt{V}}.\ \big(\lambda e \in \mathscr{B}_c.\ \sum_{e' \in \mathscr{B}_c} (\texttt{U}_{e,\,e'} \cdot \overline{\mathtt{V}}(e'))\big)$
(2) $[\![\texttt{E}]\!]^\sharp_{\overline{\mathtt{S}}} \stackrel{def}{=} \lambda\overline{\mathtt{V}}.\ (\lambda e \in \mathscr{B}_c.\ \sum_{e' \in \mathscr{B}_c} (\!|\texttt{E}_{e,\,e'}|\!)^\sharp_{\overline{\mathtt{S}}} \cdot \overline{\mathtt{V}}(e'))$*;*
(3) $[\![s_1;\, s_2]\!]^\sharp_{\overline{\mathtt{S}}} \stackrel{def}{=} [\![s_2]\!]^\sharp_{\overline{\mathtt{S}}} \circ [\![s_1]\!]^\sharp_{\overline{\mathtt{S}}}$*,*

Since $\texttt{E}_{e,\,e'}$ is always a sum of sine and cosine, $(\!|\texttt{E}_{e,\,e'}|\!)^\sharp_{\overline{\mathtt{S}}}$ can be defined using the standard interval arithmetic [MKC09].

Then, let $\texttt{M}$ be a measurement operator, the abstract semantics $[\![\texttt{M}]\!]^\sharp : \overline{\mathbb{V}}_{\mathtt{c}} \to \overline{\mathbb{D}}_{\mathtt{c}}$ is defined as:

$$[\![\texttt{M}]\!]^\sharp \stackrel{\text{def}}{=} \lambda\overline{\mathtt{V}}.(\lambda e \in \mathscr{B}_{\mathtt{c}}.\ |\overline{\mathtt{V}}(e)|^2)$$

where $|\langle [a_1, b_1], [c_1, d_1]\rangle|^2 = [a_1, b_1]^2 + [c_1, d_1]^2$, and $[a, b]^2 = [0, \max(a^2, b^2)]$ [MKC09].

Finally, the abstract semantics of $\texttt{c} := \texttt{s}; \texttt{M}$ is a function $[\![\texttt{c}]\!]^\sharp : \overline{\mathbb{S}}_{\mathtt{c}} \to (\overline{\mathbb{V}}_{\mathtt{c}} \to \overline{\mathbb{D}}_{\mathtt{c}})$ defined as:

$$[\![\texttt{c}]\!]^\sharp_{\overline{\mathtt{S}}} \stackrel{\text{def}}{=} \lambda\overline{\mathtt{V}}.\ [\![\texttt{M}]\!]^\sharp \circ [\![\texttt{s}]\!]^\sharp_{\overline{\mathtt{S}}}(\overline{\mathtt{V}}).$$

The following result tells us that computing a VQC on the abstract domain of intervals by using the so far defined abstract semantics $[\![\cdot]\!]^\sharp$ preserves all the concrete computations.

Theorem 6.1 (Soundness)**.** *The abstract VQC semantics $[\![\cdot]\!]^\sharp$ is sound. Formally, $\forall \Sigma \in \wp(\mathbb{\Sigma}_c)$ and $\forall \Psi \in \wp(\mathbb{\Psi}_c)$, then $\alpha_{\texttt{dist}}([\![\texttt{c}]\!]_\Sigma(\Psi)) \dot{\leq}_{\mathbb{R}\mathbb{I}} [\![\texttt{c}]\!]^\sharp_{\alpha_{\texttt{env}}(\Sigma)}(\alpha_{\texttt{st}}(\Psi))$*

Proof. The soundness comes directly from the fact that the abstract semantics corresponds to a composition of operations in the interval arithmetic, and these operations are all sound. □

6.3.4 An Example

We consider the same VQC as in Section 6.1. Now, suppose we want to execute the circuit for all inputs generated by perturbing the original input $[x_0, x_1]^T = [6.0, 2.7]^T$ by an $\epsilon = 0.5$.

This perturbation will create the set of environments:

$$\Sigma = \{\ \sigma \mid 5.5 < \sigma(x_0) < 6.5 \wedge 2.2 < \sigma(x_1) < 3.2 \}\,.$$

To do this, we first define the abstract environment as $\alpha_{\text{env}}(\Sigma) = \overline{\mathtt{S}} : \{x_0 \to [5.5, 6.5], x_1 \to [2.2, 3.2]\}$. Now we can execute our circuit abstractly. We start from the initial abstract state $\overline{\mathtt{V}}_0 = \{00 \to \langle [1,1],[0,0]\rangle, (01,10,11) \to \langle [0,0],[0,0]\rangle\}$, which represents the abstraction of the concrete initial state ψ_0 in Section 6.1. The first step is to execute the abstract semantics of the encoding. We need to evaluate the abstract semantics of $\mathtt{Rx}^{q_0}[x_0]$ and $\mathtt{Rx}^{q_1}[x_1]$ with $\overline{\mathtt{S}}(x_0) = [5.5, 6.5]$ and $\overline{\mathtt{S}}(x_1) = [2.2, 3.2]$, that corresponds to:

$$\overline{\mathtt{V}}_1 = [\![\mathtt{Rx}^{q_1}[x_1]]\!]^\sharp_{\overline{\mathtt{S}}} \circ [\![\mathtt{Rx}^{q_0}[x_0]]\!]^\sharp_{\overline{\mathtt{S}}}(\overline{\mathtt{V}}_0) = \left\{ \begin{array}{c} 00 \mapsto \langle [-0.454, 0.029], [0,0]\rangle \\ 01 \mapsto \langle [0,0], [-0.173, 0.049]\rangle \\ 10 \mapsto \langle [0,0], [0.824, 1.0]\rangle \\ 11 \mapsto \langle [-0.382, 0.108], [0,0]\rangle \end{array} \right\}.$$

At this point, we proceed by applying the variational part of the circuit, which computes the classification. As in the concrete case, this is obtained by applying, in sequence, the following operations $\mathtt{Ry}^{q_0}_{0.99}$, $\mathtt{Ry}^{q_1}_{-0.50}$, $\mathtt{CX}^{q_0,\,q_1}$, $\mathtt{Ry}^{q_0}_{3.27}$ and $\mathtt{Ry}^{q_1}_{-0.69}$. The final state is thus obtained by computing:

$$\begin{aligned} \overline{\mathtt{V}}_4 &= [\![\mathtt{Ry}^{q_1}_{-0.69}]\!]^\sharp_{\overline{\mathtt{S}}} \circ [\![\mathtt{Ry}^{q_0}_{3.27}]\!]^\sharp_{\overline{\mathtt{S}}} \circ [\![\mathtt{CX}^{q_0,\,q_1}]\!]^\sharp_{\overline{\mathtt{S}}} \circ [\![\mathtt{Ry}^{q_1}_{-0.50}]\!]^\sharp_{\overline{\mathtt{S}}} \circ [\![\mathtt{Ry}^{q_0}_{0.99}]\!]^\sharp_{\overline{\mathtt{S}}}(\overline{\mathtt{V}}_1) = \\ &= \left\{ \begin{array}{c} 00 \mapsto \langle [-0.159, 0.433], [-0.559, -0.355]\rangle \\ 01 \mapsto \langle [-0.404, 0.184], [0.341, 0.544]\rangle \\ 10 \mapsto \langle [-0.171, 0.328], [-0.074, 0.181]\rangle \\ 11 \mapsto \langle [-0.086, 0.413], [0.54, 0.768]\rangle \end{array} \right\}. \end{aligned} \tag{6.7}$$

The last step is the measurement. We compute $[\![\mathtt{M}]\!]^\sharp(\overline{\mathtt{V}}_4)$ obtaining the following abstract distribution:

$$\varrho^\sharp = \left\{ \begin{array}{ll} 00 \mapsto [0.126, 0.50] & 01 \mapsto [0.116, 0.460] \\ 10 \mapsto [0, 0.140] & 11 \mapsto [0.291, 0.761] \end{array} \right\} \tag{6.8}$$

As we see in Figure 6.2, we consider only the measurement of q_0. Thus, as in the concrete state, we sum the probability intervals of 00 and 10, obtaining that the probability of measuring 0 is $[0.126, 0.640]$, and we sum the probability of 01 and 11, obtaining that the probability of measuring 1 ranges between $[0.407, 1.221]$. The fact that the probability of measuring 1 exceeds 1 is due to the approximations of our abstraction.

6.4 Precision of $\mathscr{L}_{\text{VQC}}$ Abstract Semantics

In the previous section, we have introduced a *sound* abstract semantics. Soundness means that the abstract computation may add imprecision, potentially over-approximating the concrete results. In our case, this manifests as computing intervals that are larger than those we would obtain by abstracting the result of a concrete execution.

6.4.1 Sources of incompleteness

In this section, we discuss where and how the abstract computation of a VQC loses precision with respect to the concrete one. To better understand the source of imprecision, we appeal to the concept of *completeness* [CC77; CC79] of the abstraction. Analyzing completeness allows us to identify whether the loss of precision stems from the abstraction's loss of information or from the computations over the abstract domain.

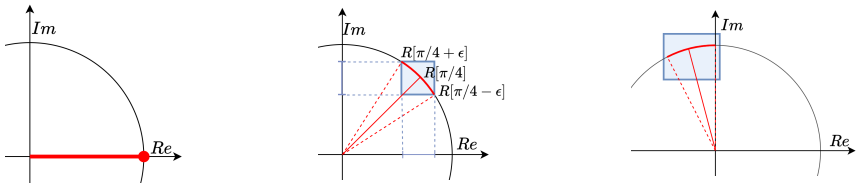
Lemma 6.1. *Let* $\mathsf{f} : \mathbb{R}^n \to \mathbb{R}$ *be defined as* $\mathsf{f}(\langle r_i \rangle_{i=1}^n) = \sum_{i=1}^n r_i$*, where* $\langle r_i \rangle_{i=1}^n$ *denotes a tuple of* n *reals. The abstraction of* f *over the interval domain is not (globally) complete.*

Proof. We proceeded to find counterexamples for completeness. Let us recall that $\mathsf{f}^\sharp : \mathbb{RI}^n \to \mathbb{RI}$ and it is defined using the interval sum. Let $X \in \wp(\mathbb{R}^n)$ be a set of tuples, we write X_j as the set $\{\ r_j \mid \forall \langle r_i \rangle_{i=1}^n \in X.\ r_j \in \langle r_i \rangle_{i=1}^n \}$, i.e., the all set elements in position j in the tuples in X. We call $\alpha_{\text{RI}^n} : \wp(\mathbb{R}^n) \to \mathbb{RI}^n$ the abstraction from a set of tuples to an interval tuple defined as: $\alpha_{\text{RI}^n}(X) = \langle \alpha_{\text{RI}}(X_j) \rangle_{j=1}^n$. Let $X = \left\{\ \langle r_i \rangle_{i=1}^n \ \middle|\ \sum_{i=1}^n r_i^2 = 1 \right\}$ a set of normalized reals tuples such that $\overline{X} = \alpha_{\text{RI}^n}(X) = \langle [-1,1] \rangle_{j=1}^n$. Thus $\mathsf{f}^\sharp(\overline{X}) = \sum_{i=1}^n [-1,1] = [-n,n]$. On the other hand, the concrete image of X through f is: $\mathsf{f}(X) = \{\ \sum_{i=1}^n r_i \mid \langle r_i \rangle_{i=1}^n \in X \}$, and by the Cauchy-Schwarz inequality and the normalization constraint: $|\sum_{i=1}^n r_i| \leq \sqrt{n} \cdot \left(\sum_{i=1}^n r_i^2\right)^{1/2} = \sqrt{n}$. Hence: $\alpha_{\text{RI}}(\mathsf{f}(X)) \leq_{\mathbb{RI}} [-\sqrt{n}, \sqrt{n}] <_{\mathbb{RI}} [-n,n] = \mathsf{f}^\sharp(\alpha_{\text{RI}^n}(X))$ i.e., the abstraction is not complete. □

From the previous lemma, we derive a more general result. The counterexample was based on tuples of normalized real numbers, i.e., $\sum r_i^2 = 1$. As shown in Equation 6.4 and Equation 6.5, the semantics of our quantum operators rely on sums of products of complex numbers. Since addition and multiplication over $\mathbb{C}$ decompose into sum of real numbers, the semantics reduces to real-valued functions over tuples of normalized values,[§] and by the previous lemma, incompleteness arises in such settings.

Theorem 6.2 (Incompleteness)**.** *The abstract semantics* $[\![\cdot]\!]^\sharp$ *of* $\mathscr{L}_{\text{VQC}}$ *is not complete.*

[§] Both state vectors ψ and matrix coefficients are normalized, due to unitarity and state normalization constraints.



(a) (b) (c)

Figure 6.3: We represent a complex number as a point with real and imaginary parts on the x and y axes. In (a) we represent $z = \langle 1, 0\rangle$; in (b) the results of computing the rotation $R[\theta](z)$ with $\theta \in [\frac{\pi}{4} - \epsilon, \frac{\pi}{4} + \epsilon]$ abstractly (blue box) and point wise (red arc); in (c) the results of the abstract and point wise application of $R[\pi/3]$ to the abstract and concrete state in (b) respectively. The results in the figures use $\epsilon \approx 0.2$.

We show incompleteness by means of the following example.

Example 6.1. Let us consider the following circuit on one qubit q: $\mathtt{s} := \mathtt{Rx}^q[x]; \mathtt{Rx}^q_{\pi/2}$, on the set of inputs $\Sigma : \{\ \sigma \mid \sigma(x) \in [\pi/2 - 0.2, \pi/2 + 0.2]\ \}$. We start from initial state $\psi_0 = \{0 \mapsto 1, 1 \mapsto 0\}$. First, we compute the abstract semantics of the first rotation gate:

$$\overline{\mathtt{V}}_1 = [\![\mathtt{Rx}^q[x]]\!]^\sharp_{\bar{\mathtt{s}}}(\alpha_{\mathtt{st}}(\{\psi_0\})) = \{0 \mapsto ([0.51, 0.86], [0, 0]), 1 \mapsto ([0, 0], [-0.86, -0.51])\}.$$

Then, applying the $\mathtt{Rx}^q_{\pi/2}$ rotation, we obtain:

$$\overline{\mathtt{V}}_f = [\![\mathtt{Rx}^q_{\pi/2}]\!]^\sharp_{\bar{\mathtt{s}}}(\overline{\mathtt{V}}_1) = \{0 \mapsto ([-0.247, 0.247], [0, 0]), 1 \mapsto ([0, 0], [-1.216, -0.722])\}.$$

We observe that abstraction introduces components with norms greater than one, since the imaginary component of the amplitude associated with basis state 1 exceeds 1, clearly values that result from overapproximation noise. Thus this example shows that, given $\psi_0 = \{0 \to 1, 1 \to 0\}$, $\alpha_{\mathtt{cI}} \circ [\![\mathtt{s}]\!]_\Sigma(\{\psi_0\}) \dot{<}_{\mathtt{cI}} [\![\mathtt{s}]\!]^\sharp_{\alpha_{\mathtt{env}}(\Sigma)}(\alpha_{\mathtt{st}}(\{\psi_0\}))$, i.e. the abstract semantics is incomplete.

We conclude that the non-relational nature of the interval domain introduces significant overapproximation, especially in the quantum setting, where amplitudes and matrix coefficients are inherently dependent due to normalization and unitary constraints.

A visual example of how overapproximation arises in the 2D complex plane is shown in Figure 6.3. We observe that applying a rotation to a point generates an arc on the unit circle. The interval abstraction, which treats each component independently, encloses this arc within a bounding box that may include points not present in the actual set. While the first abstraction in Figure 6.3b, is complete (the box exactly contains the arc without including any additional points on the circumference), the situation changes after a second rotation (Figure 6.3c): due to the extra points already introduced by the first abstraction,

the new bounding box now includes not only the concrete points in the rotated arc but also additional, spurious points along the unit circle causing incompleteness.

6.4.2 On the Completeness of the Abstract Semantics

We have seen that completeness cannot hold for all possible inputs, i.e., it fails *globally* [CC79; GRS00]. However, we can still investigate whether completeness holds under certain conditions on the operators or by restricting the set of possible inputs.[¶] This perspective allows us to better understand when and where precision is lost, namely, which inputs and which computations are responsible.

As shown in Equation 6.4 and Equation 6.5, the semantics of the statements involve sums of complex numbers that are linked by normalization constraints, which ultimately cause incompleteness. However, as we are about to show, by restricting either the type of operations or the structure of the inputs, these sums over complex numbers may simplify to single terms. This would eliminate the dependency issues discussed in the previous section and ensure completeness.

Theorem 6.3. *If* $\texttt{U}$ *is a* generalized permutation[‖]*,* $[\![\texttt{U}]\!]^\sharp$ *is complete.*

Proof. Given a generalized permutation $\texttt{U}$, for each row of the matrix, there is only one entry different from zero. We recall that given a basis $e \in \mathscr{B}_{\texttt{c}}$, the entries $\{\texttt{U}_{e,\,e'}\}_{\forall e' \in \mathscr{B}_{\texttt{c}}}$ corresponds to one row of the matrix $\texttt{U}$. Thus for each e there is one base u_e that corresponding to only non-zero entry in the row e, in other words, $\forall e \in \mathscr{B}_{\texttt{c}}.\ \texttt{U}_{e,\,e'} \neq 0$ iff $e' = u_e$. Thus, considering Equation 6.4, if $\texttt{U}$ is a generalized permutation, its semantics is $[\![\texttt{U}]\!]_\Sigma = \lambda\psi.\,(\lambda e \in \mathscr{B}_{\texttt{c}}.(\texttt{U}_{e,\,u_e} \cdot \psi(u_e))$, where u_e is the index of the only nononzerontry for each row e. Similarly, considering the Definition 6.1, we have $[\![\texttt{U}]\!]^\sharp_{\texttt{S}} = \lambda\overline{\texttt{V}}.\,(\lambda e \in \mathscr{B}_{\texttt{c}}.(\texttt{U}_{e,\,u_e} \cdot \overline{\texttt{V}}(u_e)))$. According to these simplifications, $[\![\texttt{U}]\!]^\sharp$ is complete, i.e., $\forall \Psi \in \wp(\mathbb{\Psi}_{\texttt{c}}), \Sigma \in \wp(\mathbb{\Sigma}_{\texttt{c}})\ \alpha_{\texttt{st}}([\![\texttt{U}]\!]_\Sigma(\Psi)) = [\![\texttt{U}]\!]^\sharp_{\alpha_{\texttt{env}}(\Sigma)}(\alpha_{\texttt{st}}(\Psi))$, if and only if $\alpha_{\texttt{st}}\big(\{\ \lambda e \in \mathscr{B}_{\texttt{c}}.(\texttt{U}_{e,\,u_e} \cdot \psi(u_e))\ |\ \psi \in \Psi\ \}\big) = \lambda e \in \mathscr{B}_{\texttt{c}}.(\texttt{U}_{e,\,u_e} \cdot (\alpha_{\texttt{st}}(\Psi))(u_e)$ that holds iff $\forall e \in \mathscr{B}_{\texttt{c}}.\ \alpha_{\texttt{CI}}\,(\{\ (\texttt{U}_{e,\,u_e} \cdot \psi(u_e))\ |\ \psi \in \Psi\ \}) = (\texttt{U}_{e,\,u_e} \cdot \alpha_{\texttt{CI}}\,(\{\ \psi(u_e)\ |\ \psi \in \Psi\ \}))$. Since $\texttt{U}_{e,\,u_e}$ is a number, the equality always holds. □

Corollary 6.1. $\forall k \in \mathbb{N}$, $[\![\texttt{CX}]\!]^\sharp$, $[\![\texttt{Ry}_{k\pi}]\!]^\sharp$ $[\![\texttt{Rx}_{k\pi}]\!]^\sharp$ $[\![\texttt{Rz}_k]\!]^\sharp$, *are complete.*

We now focus on the encoding operators $\texttt{E}$, which, unlike the parametric operators $\texttt{U}$, depend on the environment and thus represent families of operators. This dependency prevents us from characterizing their completeness by restricting the operator's structure alone. Instead,

[¶] When completeness is guaranteed only for specific inputs instead of universally, we refer to it as *local completeness* [Bru+23].

[‖] A generalized permutation is a matrix with exactly one non-zero entry in each row and each column.

we turn our attention to *local completeness*, which can be studied by restricting the class of possible inputs.

Theorem 6.4. $[\![\mathtt{E}]\!]^\sharp$ *is (local) complete on* $\Sigma \in \wp(\mathbb{\Sigma}_c)$ *and* $\psi \in \Psi_c$ *if* $\exists u \in \mathscr{B}_c.\ \psi(u) = 1 \wedge \forall e \neq u.\ \psi(e) = 0$, *and* $\gamma_{\text{env}} \circ \alpha_{\text{env}}(\Sigma) = \Sigma$ *namely, for any* x *the set* $\{\ \sigma(x)\ |\ \sigma \in \Sigma\ \}$ *is a closed interval.*

Proof. Let $\overline{\mathtt{S}} = \alpha_{\text{env}}(\Sigma)$. As before, by simplifying Equation 6.5 and Definition 6.1, we can say that $[\![\mathtt{S}]\!]^\sharp_{\overline{\mathtt{S}}}(\alpha_{\mathtt{st}}(\psi))$ is complete if and only if $\forall e \in \mathscr{B}_c.\ \alpha_{\mathtt{CI}}(\{\ (\!|\mathtt{E}_{e,\,u}|\!)_\Sigma \cdot \psi(u)\ |\ \sigma \in \Sigma\ \}) = (\!|\mathtt{E}_{e,\,u}|\!)^\sharp_{\overline{\mathtt{S}}} \cdot \alpha_{\mathtt{CI}}(\psi(u))$. Since $\psi(u) = 1$, the equation above holds if and only if $\alpha_{\mathtt{CI}}(\{\ (\!|\mathtt{E}_{e,\,u}|\!)_\Sigma\ |\ \sigma \in \Sigma\ \}) = (\!|\mathtt{E}_{e,\,u}|\!)^\sharp_{\overline{\mathtt{S}}}$, i.e., $(\!|\mathtt{E}_{e,\,u}|\!)^\sharp_{\overline{\mathtt{S}}}$ is complete. From subsection 6.3.3, we know that $(\!|\mathtt{E}_{e,\,u}|\!)^\sharp_{\overline{\mathtt{S}}}$ is defined as the *bca* of $(\!|\mathtt{E}_{e,\,u}|\!)_\Sigma$. Therefore, if $\gamma_{\text{env}} \circ \alpha_{\text{env}}(\Sigma) = \Sigma$, then $(\!|\mathtt{E}_{e,\,u}|\!)^\sharp_{\overline{\mathtt{S}}}$ is trivially complete [CC79; GRS00]. □

Finally, we consider when local completeness holds for the final measurement.

Theorem 6.5. *If* $\forall e \in \mathscr{B}_c.\ \mathrm{Re}(\Psi_c(e)) = 0 \vee \mathrm{Im}(\Psi_c(e)) = 0$, $[\![\mathtt{M}]\!]^\sharp$ *is (local) complete on* Ψ.

Proof. By definition, if for all $\psi \in \Psi$ and for all bases e, $\mathrm{Re}(\Psi_c(e)) = 0 \vee \mathrm{Im}(\Psi_c(e)) = 0$, the measurement operators simply compute the square of a real number. The abstraction of the square function is always complete in real interval arithmetic by definition [MKC09]. □

6.5 (Abstract) VQC-based classifier

Similarly to what happens when dealing with NNs, VQCs are primarily employed for data classification tasks [WTD24]. In the previous section, we discussed how to abstract the execution of a VQC when its input is represented as an interval. In this section, we exploit both the concrete and abstract semantics of VQC for modeling the final step of its employment as a classifier: the classification function, i.e., the function deputed to map the output of the quantum computation to a corresponding class label.

6.5.1 VQC-based Classification

A VQC returns a probability distribution over the set of possible results. Crucially, for classification purposes, the output distribution of a VQC must be post-processed. As illustrated in the example in Section 6.1, classification is determined by the value of the first qubit. This means that, for instance, the probabilities of measuring 10 and 00 are aggregated into one class, while 11 and 01 are aggregated into another. More generally, the number of target classes is often smaller than the number of possible results, requiring a

mapping from quantum outputs to class labels by summing the probabilities of grouped outcomes accordingly.

For a $\mathbf{c} \in \mathscr{L}_{\texttt{vqc}}$, we introduce the function $\xi : \mathbb{\Delta}_{\mathbf{c}} \times \bigcup_{n \leq |\texttt{QV}_{\mathbf{c}}|} \texttt{QV}_{\mathbf{c}}^{n} \to \bigcup_{n \leq |\texttt{QV}_{\mathbf{c}}|} \{0,1\}^n$ that determines which class is the selected one, based on the outcome of the VQC execution. Let $\rho \in \mathbb{\Delta}_{\mathbf{c}}$ be the resulting distribution of a VQC, and let $\mathbf{q} \in \texttt{QV}_{\mathbf{c}}^{n}$ be a tuple of n qubits, we define:

$$\xi(\rho, \mathbf{q}) \stackrel{\text{def}}{=} \arg \max_{b \in \{0,1\}^n} \sum_{e \in \mathscr{B}_{\mathbf{c}}. e[\mathbf{q}]=b} \rho(e),$$

where $e[\mathbf{q}]$ denotes the projection of the bitstring e onto the positions in $\mathbf{q}$.
As an example, consider $\mathbf{q} = \langle q_0 \rangle$, meaning that we are interested in the classification task observing the value of the single ($n = 1$) qubit q_0. Let us consider the distribution resulting from the VQC in Section 6.1, i.e., $\rho = \{00 \mapsto 0.26,\ 01 \mapsto 0.21,\ 10 \mapsto 0.01,\ 11 \mapsto 0.52\}$, where the first bit corresponds to qubit q_1 and the second to q_0. In order to determine the selected class, we compute the aggregated probabilities: for $b = 0$, i.e., outcomes where $q_0 = 0$, the relevant bases are 00 and 10 ($00[\mathbf{q}] = 10[\mathbf{q}] = 0$), with $\rho(00) = 0.26$ and $\rho(10) = 0.01$, hence $\rho(10) + \rho(00) = 0.27$. Similarly, for $b = 1$, i.e., outcomes where $q_0 = 1$, the relevant bases are 01 and 11 ($01[\mathbf{q}] = 11[\mathbf{q}] = 1$), with $\rho(01) = 0.21$ and $\rho(11) = 0.52$, so $\rho(01) + \rho(11) = 0.73$. Since $0.73 > 0.27$, we have: $\xi(\rho, \langle q_0 \rangle) = 1$. That is, the most likely class according to the classification rule based on q_0 is the one associated with 1.

We can now define the semantics of the whole VQC classifier. Without loss of generality, we assume that the execution of a VQC always starts from a specific initial state where all quantum variables are in the value 0. Thus, given $\mathbf{c} \in \mathscr{L}_{\texttt{vqc}}$ ($|\texttt{QV}_{\mathbf{c}}| = n$), we define the initial state $\psi_0 \in \mathbb{\Psi}_{\mathbf{c}}$ as:

$$\psi_0 \stackrel{\text{def}}{=} \lambda e \in \mathscr{B}_{\mathbf{c}}. \begin{cases} 1 & e = 0^n \\ 0 & \text{othewise} \end{cases} \tag{6.9}$$

If qubits $\mathbf{q} \in \texttt{QV}_{\mathbf{c}}^{n}$ ($n \leq |\texttt{QV}_{\mathbf{c}}|$) are the qubits encoding the classical dataset to be classified, the semantics of the whole VQC classification algorithm can be modeled as $\xi(\llbracket \mathbf{c} \rrbracket_{\sigma}(\psi_0), \mathbf{q})$. We abuse notation by using the function ξ also to denote its additive lift to sets of distributions:

$$\xi(\llbracket \mathbf{c} \rrbracket_{\Sigma}(\psi_0), \mathbf{q}) \stackrel{\text{def}}{=} \{\ \xi(\llbracket \mathbf{c} \rrbracket_{\sigma}(\psi_0), \mathbf{q})\ |\ \sigma \in \Sigma\ \}\,. \tag{6.10}$$

An explicit definition of the classification functions is given in [Sch+20], where the authors define a classification function based on the measurement of a single qubit followed by classical post-processing with a tunable bias. This bias, implemented purely at the classical level by adjusting the decision threshold, allows the model to learn more flexible decision boundaries without increasing the quantum circuit depth. For the sake of simplicity but without loss of

generality, we omit such bias terms in this chapter. Nonetheless, our framework is general enough to accommodate them.

6.5.2 Abstract VQC-based classification

The classification process described above, which assigns a unique class based on the output distribution, applies to concrete executions of the VQC. However, when abstracting the semantics of the VQC, the output is no longer a single probability distribution but an abstract distribution, where each standard basis vector is associated with an interval of possible probabilities. As a consequence, the aggregated probabilities for different classes may result in overlapping intervals, making it impossible to determine a unique selected class with certainty.

A first step toward managing the uncertainty introduced by abstraction is to return a *set* of potential selected classes, i.e., all classes that could be chosen given the overlapping probability intervals. This allows us to assess whether all possible classifications are acceptable, or a refinement of the abstraction is necessary to reduce ambiguity, as it has been formalized for verifying abstract robustness in NN [GMP24; MMF25].

Hence, we abstract the ξ function in order to select only the intervals with a lower bound greater than any other interval distribution. Specifically, $\xi^\sharp : \varrho^\sharp \times \bigcup_{n \leq |\mathtt{QV_c}|} \mathtt{QV}_\mathtt{c}^n \to \bigcup_{n \leq |\mathtt{QV_c}|} \wp(\{0,1\}^n)$ selects the set of potential classes with higher probability. Formally, given $\mathbf{q} \in \mathtt{QV}_\mathtt{c}^n$

$$\xi^\sharp(\varrho^\sharp, \mathbf{q}) \stackrel{\text{def}}{=} \left\{ b \in \{0,1\}^n \;\middle|\; \nexists b' \in \{0,1\}^n . \sum_{e \in \mathscr{B}_\mathtt{c} . e[\mathbf{q}] = b} \varrho^\sharp(e) <_{\mathbb{R}\mathbb{I}} \sum_{e \in \mathscr{B}_\mathtt{c} . e[\mathbf{q}] = b'} \varrho^\sharp(e) \right\},$$

where the summation $\sum$ is computed according to the real interval arithmetic. Since the image of the function is in $\wp(\{0,1\}^n)$ and we are considering abstracted distributions, we may have an overlap of classes that can be selected. To see this on an example, let us consider the following abstract distribution: $\varrho^\sharp = \{000 \to [0.2, 0.46], 001 \to [0, 0.2], 010 \to [0.1, 0.03], 101 \to [0.11, 0.6], 111 \to [0.1, 0.6]\}$ on three qubits $\langle q_2, q_1, q_0 \rangle$ standard basis vectors, where we omit the ones mapped to the singleton $[0, 0]$. Let $\mathbf{q} = \langle q_2, q_0 \rangle$, we first compute $\sum_{e \in \mathscr{B}_\mathtt{c} . e[\mathbf{q}] = b} \varrho^\sharp(e)$ and we obtain: $[0.29, 0.56]$ for 00, $[0, 0.2]$ for 01, $[0.22, 1.2]$ for 11 and $[0, 0]$ for 10.** In this case, there is no single class whose lower bound exceeds the upper bounds of all other classes. However, we can observe that the configurations $\{01, 10\} \in \wp(\{0,1\}^2)$ have definitively lower probabilities than $\{00, 11\} \in \wp(\{0,1\}^2)$. Therefore, the

** For the sake of readability, we assume that the result represents a map from the qubits of interest to their values, i.e., if $\mathbf{q} = \langle q_2, q_0 \rangle$ then 01 stands for $q_2 \mapsto 0$ and $q_0 \mapsto 1$.

function $\xi^\sharp$ will return the latter set of configurations, whose probability intervals overlap but are still greater than those of the others.

Hence, given $\mathbf{q} \in \mathtt{QV}_c^n$ $(n \leq |\mathtt{QV}_c|)$ consisting of the list of qubits to consider for the classification, and the abstract initial state $\overline{\mathtt{V}}_0 = \alpha_{\mathtt{st}}(\{\psi_0\})$, the abstract semantics of the classification algorithm is:

$$\xi^\sharp([\![\mathtt{c}]\!]^\sharp_{\mathtt{s}}(\overline{\mathtt{V}}_0), \mathbf{q}),$$

whose task consists of computing such an approximated classification, potentially returning a *set* of possible selected results. Then we can observe that abstract classification is also sound.

Proposition 6.5. $\forall \Sigma \in \wp(\mathbb{\Sigma}_c), \mathbf{q} \in \mathtt{QV}_c^n$ $(n \leq |\mathtt{QV}_c|)$. $\xi([\![c]\!]_\Sigma(\psi_0), \mathbf{q}) \subseteq \xi^\sharp([\![c]\!]^\sharp_{\alpha_{\mathtt{env}}(\Sigma)}(\overline{\mathtt{V}}_0), \mathbf{q})$, *i.e., the abstract classification is sound.*

6.5.3 Robustness Verification of (Abstract) VQC Classifier

In this section, we address the robustness verification problem, sometimes referred to as *stability*, for VQCs, which concerns verifying that small perturbations in the input do not alter the original output classification. The formalization presented below can be naturally extended to other properties, such as safety, in analogy with classical neural network verification approaches [Liu+21]. Specifically, we focus on input perturbations bounded by an ℓ_∞-norm centered at a nominal input $x \in \mathbb{R}^n$, formally defined as:

$$\mathscr{C}_\infty(x, \varepsilon) = \{x' \in \mathbb{R}^n \mid \|x' - x\|_\infty \leq \epsilon\}.$$

In practice, we analyze the robustness of the circuit on abstract input intervals in the form:

$$\Sigma_{\sigma,\epsilon} \stackrel{\text{def}}{=} \{\ \sigma' \mid ||\sigma(x) - \sigma'(x)||_\infty \leq \epsilon\},$$

where σ is the perturbated input (environment) and ϵ represents the noise applied to such input. We now formally define the robustness verification problem for VQC-based classifiers.

Definition 6.2 (Robustness Verification of VQC (RVVQC))**.** *Let $\mathtt{c} \in \mathscr{L}_{\mathtt{VQC}}$, $\epsilon \in \mathbb{R}$ be the input perturbation, and $\mathbf{q} \in \mathtt{QV}_c^n$ $(n \leq |\mathtt{QV}_c|)$ be the list of output observed qubits. We say that $\mathtt{c}$ is robust w.r.t. $\mathbf{q}$ and ϵ (denoted $\mathtt{c} \models_{\mathbf{q},\epsilon}$ RVVQC) iff $\forall \sigma \in \mathbb{\Sigma}_c$ it holds*

$$\xi([\![c]\!]_{\Sigma_{\sigma,\epsilon}}(\psi_0), \mathbf{q}) \subseteq \{\xi([\![c]\!]_\sigma(\psi_0), \mathbf{q})\}. \tag{6.11}$$

This definition implies that a VQC classifier for $\mathtt{c}$ is robust if, when executed on all perturbed versions of the input σ, it consistently produces the same output class as when executed on

σ itself. In other words, the set of output classes obtained under all admissible perturbations of σ must be a singleton containing only the original classification.

We now focus our attention on the computability and computational complexity aspects of the VQC verification problem. We first observe that, in the RVVQC problem, the input set of environments has the same cardinality as the real numbers, since each environment corresponds to a possible mapping between a finite set of variables and real values ($\mathbb{\Sigma}_{\mathtt{c}} \stackrel{\text{def}}{=} \mathtt{CV_c} \to \mathbb{R}$, see subsection 6.2.2). Consequently, the verification problem (and its dual, the confutation problem) is, in general, non-computable. This non-computability arises independently of whether the property to be verified on each individual input is recursive (i.e., decidable in finite time). This means that, in order to computationally treat the verification problem, we need to restrict the input space to *computable real numbers with a fixed maximal precision* only, namely those reals that can be computed, to within any desired precision, by a finite, terminating algorithm, denoted by $\overline{\mathbb{R}}$.[††] This makes the input set space ($\mathbb{\Sigma}_{\mathtt{c}}$) a countable subset of reals [Min67] and $\Sigma_{\sigma,\epsilon}$ discrete and finite, making Equation 6.11 decidable. In the following part of the paper, we consider as input environments only those defined by mapping variables to computable and discrete reals.

Proposition 6.6. *Let* $\mathtt{c} \in \mathscr{L}_{\mathtt{VQC}}$, $\epsilon \in \overline{\mathbb{R}}$, *and* $\mathbf{q} \in \mathtt{QV}_{\mathtt{c}}^{n}$ $(n \leq |\mathtt{QV}_{\mathtt{c}}|)$

(1) $\mathtt{CP} \stackrel{def}{=} \{\ \mathtt{c} \mid \mathtt{c} \not\models_{\mathbf{q},\epsilon} \mathit{RVVQC}\}$ *is computable (recursive enumerable).*
(2) $\mathtt{VP} \stackrel{def}{=} \{\ \mathtt{c} \mid \mathtt{c} \models_{\mathbf{q},\epsilon} \mathit{RVVQC}\}$ *is not computable;*

The intuition is that, for computing VP we have to execute the VQC on an infinite (even if computable) set of inputs, while for computing CP it is sufficient to find one input not satisfying Equation 6.11, and if it exists, we can find it in finite time over a discrete space of inputs (e.g., by using a technique called *dovetail* for searching on the input space without diverging).

If we restrict to a finite set of inputs (i.e., environment), then both problems collapse to recursive problems. Hence, consider $\mathbb{\Sigma}_{\mathtt{c}}^{\mathtt{f}} \stackrel{\text{def}}{=} \mathtt{CV_c} \to R$, with $R \subset_{\mathtt{f}} \overline{\mathbb{R}}$ a finite subset of reals. In this case, robustness verification in Definition 6.2 is denoted $\mathtt{c} \models_{\mathbf{q},\epsilon} \mathtt{f}$-RVVQC. Nevertheless, even when made decidable, providing an answer to the f-RVVQC problem requires testing the VQC on a potentially huge set of inputs (depending on the chosen precision in approximating reals), which is clearly hard.

Theorem 6.6. $\mathtt{VP}^{\mathtt{f}} \stackrel{def}{=} \{\ \mathtt{c} \mid \mathtt{c} \models_{\mathbf{q},\epsilon} \mathtt{f}\text{-}\mathit{RVVQC}\}$ *is recursive but* $\mathtt{c} \models_{\mathbf{q},\epsilon} \mathtt{f}\text{-}\mathit{RVVQC}$ *is NP-hard.*

[††] A computable number [is] one for which there is a Turing machine which, given n on its initial tape, terminates with the nth digit of that number [encoded on its tape] [Min67].

Proof. To show the hardness of the problem, we show the following reduction 3-SAT $\preccurlyeq_k$ RVVQC. In particular, we will show that any 3-SAT instance on a Boolean formula Φ can be converted into a VQC $\mathtt{c}$ such that the property Π is satisfied on $\mathtt{c}$ if and only if Φ is satisfiable.

Let $\Phi = (C_1 \wedge \ldots \wedge C_m)$ be a 3-conjunctive normal form (CNF) over a set of Boolean variables $X = \{x_1, x_2, \ldots, x_n\}$. Each formula is a conjunction of clauses, where each clause is a disjunction of exactly three literals. Each literal is either a variable x_k or its negation $\neg x_k$, for some $k \in \{1, \ldots, n\}$. An assignment $a : X \to \{0, 1\}$ satisfies Φ if it satisfies all the clauses simultaneously.

For the reduction, we first note that Φ can be viewed as a Boolean circuit which computes a function $f : a \to \{0, 1\}$, such that $f(a) = 1$ if and only if a satisfies Φ. Hence, from [KSV02, Section 7] we can transform f into a quantum circuit $\mathtt{U}_{3-\mathrm{SAT}}$ acting on poly(n) qubits. However, for the complete reduction, since we aim to obtain a VQC from Φ, we first need to introduce an *encoding gadget* $\mathtt{E}$ which takes as input a and encodes it for the $\mathtt{U}_{3-\mathrm{SAT}}$. In Figure 6.4 we report a schematic overview of $\mathtt{E}$.

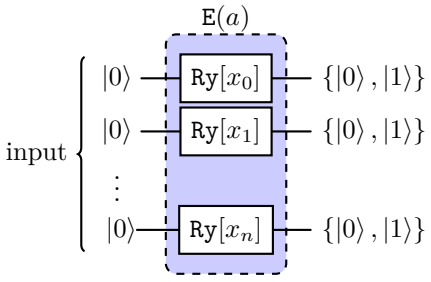


Figure 6.4: *Encoding Gadget* $\mathtt{E}$

In detail, the input of $\mathtt{E}$ corresponds to $|X|$ qubits initialized to $|0\rangle$. Then for a given an assignment a for $\{x_1 \to \{0, 1\}, \ldots, x_n \to \{0, 1\}\}$, we define a static environment σ_a as:

$$\sigma_a(x_i) \leftarrow \begin{cases} \pi & \text{if } a(x_i) = 1 \\ 0 & \text{if } a(x_i) = 0 \end{cases} \tag{6.12}$$

For the output of $\mathtt{E}$, we then prove the following.

Lemma 6.2. *For any assignment $a : X \to \{0, 1\}$, the evaluation of the encoding gadget $\mathtt{E}$ w.r.t. the static environment σ_a produces an output which is consistent with a.*

Proof. The proof is straightforward by observing that if the original assignment for an x_i is

1, then the gate rotation $\texttt{Ry}[\sigma_a(x_i) = \pi]$ will produce $|1\rangle$ as output. On the other hand, if $x_i = 0$ then $\texttt{Ry}[\sigma_a(x_i) = 0]$ results in the identity leaving the state as $|0\rangle$. □

By combining the *encoding gadget* $\texttt{E}$ with the $\texttt{U}_{3-\text{SAT}}$ circuit, we obtain a VQC $\texttt{c}$ as desired. Clearly, the reduction from any 3-SAT instance Φ to the VQC $\texttt{c}$ can be carried out in polynomial time. Let Φ have n variables and m clauses. The encoding gadget $\texttt{E}$ introduces one R_y rotation gate per variable, thus requiring $O(n)$ gates. For each clause, the construction of the $\texttt{U}_{3\text{-SAT}}$ circuit involves a constant number of controlled-NOT and negation gates (to implement 3-literal OR via De Morgan), requiring $O(1)$ gates per clause. The final conjunction over the clause outputs requires $m - 1$ additional controlled operations. Hence, the total number of quantum gates and qubits used in $\texttt{c}$ is $O(n + m)$.

To complete the proof, we need to show the following.

Lemma 6.3. *Any 3-SAT formula Φ can be reduced into a VQC $\texttt{c}$ and a property $\Pi = \{1\}$, such that*
Π is satisfiable on $\texttt{c}$ if and only if Φ is satisfiable.

Proof. Without loss of generality, we will prove the lemma exploiting the example of the reduction of a simple formula Φ with $n = 3$ variables and $m = 2$ clauses into a VQC $\texttt{c}$ reported in Figure 6.5.

$\boxed{\Longrightarrow}$

We need to show that if $\xi([\![\texttt{c}]\!]_{\sigma_a}(\psi_0), \mathbf{q}) \subseteq \Pi$, i.e., the concrete execution of the VQC with the environment σ_a, then there exists an assignment a such that $\Phi(a) = 1$. Assume that $\xi([\![\texttt{c}]\!]_{\sigma_a}(\psi_0), \mathbf{q}) \subseteq \Pi = \{1\}$. This means that when the VQC $\texttt{c}$ is executed and measured, the output qubit is observed in the state 1. By construction, $\texttt{c}$ outputs $|1\rangle$ if and only if all clause qubits a_i (for $i = 1, \ldots, m$) are set to $|1\rangle$, meaning that each corresponding clause of the formula Φ is satisfied. Therefore, measuring the output qubit in state $|1\rangle$ implies that the assignment used by the circuit satisfies every clause in Φ. By Lemma 6.2, each assignment a to the Boolean variables of Φ corresponds to a valid and consistent quantum state prepared by the encoding gadget $\texttt{E}$. Hence, if the output qubit of $\texttt{c}$ is measured as $|1\rangle$, then there must exist at least one assignment a such that $\Phi(a) = 1$, which means that Φ is satisfiable.

$\boxed{\Longleftarrow}$

Fix an assignment a for $\Phi(x_1, x_2, x_3)$, such that $\Phi(a) = 1$. The first step in the reduction is to produce the output of the *encoding gadget* $\texttt{E}$, for the assignment a, i.e., $\texttt{E}(a)$. From

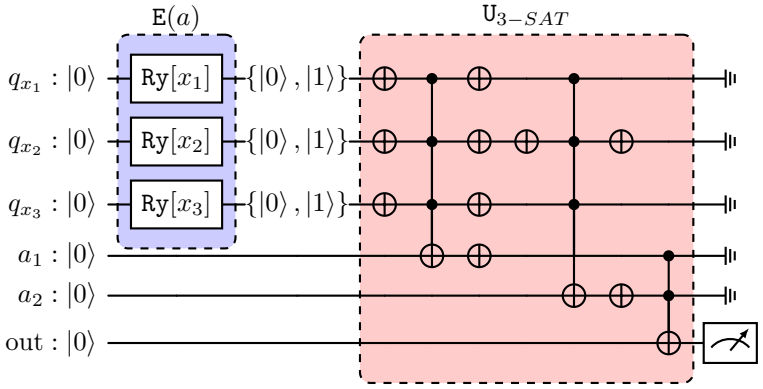


Figure 6.5: Complete reduction of a formula $\Phi(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$ into a VQC c.

Lemma 6.2 we know that this gadget will produce an output consistent with a, for instance if the assignment a produce $\{x_1 \to 0, x_2 \to 1, x_3 \to 1\}$ then the corresponding output of $\mathtt{E}(a)$ which becomes the input for $\mathtt{U}_{3-\mathrm{SAT}}$ will be $\{|0\rangle, |1\rangle, |1\rangle\}$. The $\mathtt{U}_{3-\mathrm{SAT}}$ circuit, first applies the negation gates ($\oplus$) to $q_{x_1}, q_{x_2}, q_{x_3}$ to get their negations. By applying a multi-controlled NOT, it computes $(\neg q_{x_1} \wedge \neg q_{x_2} \wedge \neg q_{x_3})$, storing the result in a_1. By negating the result of $a_1 = \neg(\neg q_{x_1} \wedge \neg q_{x_2} \wedge \neg q_{x_3}) = (q_{x_1} \vee q_{x_2} \vee q_{x_3})$ it encodes the OR operation for the first clause, as performed in Φ. Similarly, it is done for the second clause, so the circuit first restores the original values of $q_{x_1}, q_{x_2}, q_{x_3}$ by performing a negation on all the values. It then compute the negation only on q_{x_2}, storing the result in $a_2 = (x_1 \wedge \neg x_2 \wedge x_3)$. After this, it exploits again a multi-controlled NOT to get the result of $a_2 = \neg(x_1 \wedge \neg x_2 \wedge x_3) = (\neg x_1 \vee x_2 \vee \neg x_3)$. Finally, to encode the AND of the clauses, it simply performs the $\oplus$ operation on a_1 and a_2, storing the final result in $out = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$, which is precisely the original formula Φ. Hence if a satisfies Φ then it also necessarily holds by construction that $a_1, a_2 = |1\rangle$, which once measured, will produce $1 \subseteq \Pi$.

□

The proof is complete.

□

We proved that enumerating the safe inputs of a VQC is NP-hard. Now, we investigate whether our abstract interpretation framework can, in some sense, mitigate the inherent complexity of the problem. From Proposition 6.5, we know that classification based on the abstract semantics of a VQC is sound. This soundness ensures conservativeness with respect to the concrete computations—that is, no concrete behaviors are missed, and thus

no possible classifications are lost. Hence, we can use the computed over-approximation in order to verify a desirable property $\mathtt{f}$-RVVQC. Specifically, if the abstract output assigns a strictly higher probability to one class compared to all others, then—by the soundness of the abstraction—that class is guaranteed to be selected in the concrete execution as well.

Proposition 6.7. *Consider a VQC* $\mathtt{c}$ *and* $\mathbf{q} \in \mathtt{QV}_c^n$ *(*$n \leq |\mathtt{QV}_c|$*). Given the initial abstract state* $\overline{\mathbb{V}}_0$*. Then* $\mathtt{c}$ *is robust iff* $\forall \sigma \in \mathbb{\Sigma}_c^{\mathtt{f}}$ *it holds*

$$\xi^\sharp([\![c]\!]^\sharp_{\alpha_{\mathsf{env}}(\Sigma_{\sigma,\epsilon})}(\overline{\mathbb{V}}_0), \mathbf{q}) \subseteq \{\xi([\![c]\!]_\sigma(\psi_0), \mathbf{q})\}.$$

This result shows that when the over-approximation introduced by abstraction does not yield overlapping probability intervals across classes, the outcome provides a precise robustness certificate (Definition 6.2). In other words, the abstract computation is sound and complete with respect to the VQC's behavior. In the worst case, the abstraction may introduce an over-approximation that causes the probability intervals of different classes to overlap, leading to ambiguous classifications and inconclusive verification results. Nonetheless, as in classical NN-Verification, abstract interpretation plays a crucial role in providing *provable guarantees* about the classifier's behavior. Through this framework, we can formally reason about the sources and impact of imprecision, which represent the primary source of uncertainty in approximation-based analyses. In this context, managing precision becomes a central challenge. By identifying where imprecision arises, it becomes possible to strike a balance between uncertainty and computational complexity, achieving a trade-off that preserves both efficiency and reliability in verification.

6.6 Recovering Precision in Abstract VQC Verification

As discussed in the previous section, the interval-based abstract semantics is sound but sometimes incomplete, leading to a potential loss of precision. In this section, we introduce techniques to mitigate the loss of precision resulting from this approach.

Clipping the Intervals. In Example 6.1, we observe that the abstraction introduces components with norms greater than one. The problem is in the imaginary part of the amplitude associated with the basis 1 exceeding 1, due to the overapproximation error. To mitigate this issue, a possible solution is to remove such invalid elements by clipping all intervals to lie within $[-1, 1]$. That is, at each step of the computation, we intersect each component of the abstract state with the interval $[-1, 1]$. Formally, we define a function $\mathrm{clip}_{\mathtt{CI}} : \overline{\mathbb{V}}_{\mathtt{c}} \to \overline{\mathbb{V}}_{\mathtt{c}}$ as $\mathrm{clip}_{\mathtt{CI}} \stackrel{\text{def}}{=} \lambda\overline{\mathbb{V}}.\,(\lambda e \in \mathscr{B}_{\mathtt{c}}.\langle \mathrm{Re}(\overline{\mathbb{V}}(e)) \cap [-1,1], \mathrm{Im}(\overline{\mathbb{V}}(e)) \cap [-1,1]\rangle)$. A 2D illustration is shown in Figure 6.6a.

Proposition 6.8. *The clip operation is sound for all quantum statements, i.e.,* $\forall \Sigma \in \wp(\mathbb{\Sigma}_c)$, $\alpha_{\mathtt{CI}}([\![s]\!]_\Sigma(\psi_0)) \dot{\leq}_{\mathbb{CI}} clip_{\mathtt{CI}}([\![s]\!]^\sharp_{\alpha_{\mathtt{CI}}(\Sigma)}(\overline{\mathbb{V}}_0))$

Proof. The soundness follows from that given a $\psi \in [\![\mathsf{s}]\!]_\Sigma(\psi_0)$, $\forall e \in \mathscr{B}_\mathsf{c}$, $-1 \geq \mathrm{Re}(\psi(e)) \geq 1$ and $-1 \geq \mathrm{Im}(\psi(e)) \geq 1$. Thus the elements eliminated by the $\mathrm{clip}_{\mathtt{CI}}$ are not in $\alpha_{\mathtt{CI}}([\![\mathsf{s}]\!]_\Sigma(\psi_0))$. □

Similarly, after performing a measurement, the resulting probability intervals may have upper bounds exceeding 1. Since measurement outcomes must lie within the range $[0, 1]$, we can define another function that applies a clipping by intersecting the interval with $[0, 1]$. Formally, we can define a function $\mathrm{clip}_{\mathtt{RI}} : \overline{\mathbb{D}}_\mathsf{c} \to \overline{\mathbb{D}}_\mathsf{c}$ as $\mathrm{clip}_{\mathtt{RI}} \stackrel{\mathrm{def}}{=} \lambda \varrho^\sharp. (\lambda e \in \mathscr{B}_\mathsf{c}. \varrho^\sharp(e) \cap [0, 1])$. For similar consideration as in Proposition 6.8, we have

Proposition 6.9. *The clip operation on the measurement is sound.*

Since both clip functions are sound, we can guarantee that applying the clip operation at each step of the abstract semantics is sound.

'Symbolic' Execution of Parametric Circuit. Another technique employed in classical NN-Verification for reducing the overapproximation is called *symbolic interval propagation* [Wan+18]. At a high level, the idea is to track dependencies between variables symbolically, during abstract computation, reducing overapproximation compared to naive interval propagation methods. However, its precision drops in the presence of non-convex transformations, and it often requires interval concretization to remain tractable.

Inspired by this approach, we investigate whether this simple yet effective technique can also be applied in a quantum setting. We begin by observing that, due to the linearity of the parametric part of the VQC, the evolution of a concrete quantum state follows the standard

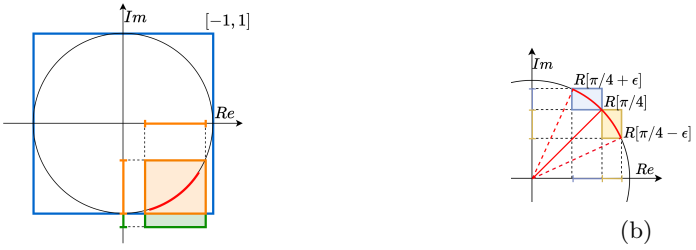


Figure 6.6: An example of the clipping function (a), and of the refinement based on splitting intervals (b)

rule of operator composition, i.e., matrix multiplication. Specifically, given a vector state $|\psi\rangle$ and two unitary operators $\mathtt{U}$ and $\mathtt{U}'$, $\mathtt{U}' \cdot (\mathtt{U} \cdot |\psi\rangle) = (\mathtt{U}' \cdot \mathtt{U}) \cdot |\psi\rangle$. On the other hand, when working with abstract states, given an abstract state $\overline{\mathtt{V}}$ and two operations $\mathtt{U}$ and $\mathtt{U}'$, the abstract semantics satisfy: $[\![\mathtt{U}' \cdot U]\!]^\sharp(\overline{\mathtt{V}}), \dot{\leq}_{\mathbb{C}\mathbb{I}} [\![\mathtt{U}']\!]^\sharp \circ [\![U]\!]^\sharp(\overline{\mathtt{V}})$. This inclusion reflects the fact that by composing $\mathtt{U}$ and $\mathtt{U}'$ first, we can perform the concrete operations exactly (without overapproximation), and only afterwards apply a single abstract operation to the result, thus reducing the accumulation of overapproximation errors in the analysis.

Example 6.2. In order to understand the idea in practice, consider the example in subsection 6.3.4. In this case, if we first compute the full operator

$$O \stackrel{\text{def}}{=} \mathtt{Ry}^{q_1}_{-0.69} \cdot \mathtt{Ry}^{q_0}_{3.27} \cdot \mathtt{CX}^{q_0,\, q_1} \cdot \mathtt{Ry}^{q_1}_{0.99} \cdot \mathtt{Ry}^{q_0}_{-0.50},$$

and then apply it to the initial abstract state, we can directly compute the final distribution as:

$$\varrho^\sharp_s = [\![\mathtt{M}]\!]^\sharp \circ [\![O]\!]^\sharp_{\mathbb{S}}(\overline{\mathtt{V}}_1) = \left\{ \begin{matrix} 00 \mapsto [0.165, 0.394] & 01 \mapsto [0.128, 0.410] \\ 10 \mapsto [0, 0.068] & 11 \mapsto [0.320, 0.698] \end{matrix} \right\}. \tag{6.13}$$

Thus, we find q_0 equals to 0 w.p. $[0.165, 0.462]$ and equals to 1 w.p. $[0.448, 1.108]$. We clearly observe that the intervals are narrower compared to those in $\varrho^\sharp$ from Equation 6.8, which was obtained by applying the operators step by step. However, we are currently unable to verify that the VQC is robust with respect to the initial input $x = [6, 2.7]$ perturbed by an $\epsilon = 0.5$. We can improve the abstraction process by introducing another technique, called iterative refinement.

Iterative Refinement. In classical NNs, one common technique to improve the precision of interval analysis is to recursively split the input interval into smaller sub-intervals [Wan+18]. For instance, analyzing the network over a broad input range such as $[0, 10]$ might produce a loose output bound like $[2, 30]$. However, if we divide the input into two smaller intervals, say $[0, 5]$ and $[5, 10]$, and analyze each one separately, we might obtain tighter bounds such as $[2, 12]$ and $[10, 20]$, respectively. Taking the union of these results yields a refined output bound of $[2, 20]$, improving the overall precision of the analysis. This strategy helps reduce overestimation caused by input dependencies. Its effectiveness is supported by the fact that NNs are Lipschitz continuous, which ensures that repeated input refinement will eventually converge to arbitrarily tight output bounds within a finite number of steps.

In our setting, the encoding functions used to generate quantum states, such as sine and cosine, are continuous. This continuity allows us to adopt a similar strategy; by partitioning the input domain and analyzing smaller subregions independently, we can derive more precise output intervals. In Figure 6.6b, we provide a 2D visualization of this idea. Instead

of computing the abstract rotation $\mathtt{R}[\pi/4-\epsilon, \pi/4+\epsilon]$ all at once, we split the input interval and compute separately $\mathtt{R}[\pi/4-\epsilon, \pi/4]$ and $\mathtt{R}[\pi/4, \pi/4+\epsilon]$. As shown in the figure, the union of these two results yields a tighter overapproximation than applying the rotation over the entire interval at once.

Once again, to provide a practical example of this technique, let us consider again the VQC $\mathtt{c}$ used in subsection 6.3.4. To further improve precision, we can refine the input domain by splitting one of the two input intervals, say, x_0. We thus define two sub-environments: $\overline{\mathtt{S}}_0 \colon \{x_0 \mapsto [5.5, 6],\ x_1 \mapsto [2.2, 3.2]\}$, $\overline{\mathtt{S}}_1 \colon \{x_0 \mapsto [6, 6.5],\ x_1 \mapsto [2.2, 3.2]\}$, such that $\overline{\mathtt{S}} = \overline{\mathtt{S}}_0 \cup \overline{\mathtt{S}}_1$. Executing the abstract semantics on these two environments, using symbolic execution, we obtain:

$$[\![\mathtt{c}]\!]^{\sharp}_{\overline{\mathtt{S}}_0} = \left\{ \begin{matrix} 00 \mapsto [0.176, 0.387] & 01 \mapsto [0.139, 0.355] \\ 10 \mapsto [0, 0.057] & 11 \mapsto [0.322, 0.674] \end{matrix} \right\},$$

$$[\![\mathtt{c}]\!]^{\sharp}_{\overline{\mathtt{S}}_1} = \left\{ \begin{matrix} 00 \mapsto [0.195, 0.285] & 01 \mapsto [0.148, 0.367] \\ 10 \mapsto [0, 0.048] & 11 \mapsto [0.40, 0.635] \end{matrix} \right\}.$$

Taking the join, we obtain:

$$[\![\mathtt{c}]\!]^{\sharp}_{\overline{\mathtt{S}}_0} \cup [\![\mathtt{c}]\!]^{\sharp}_{\overline{\mathtt{S}}_1} = \left\{ \begin{matrix} 00 \mapsto [0.176, 0.387] & 01 \mapsto [0.139, 0.367] \\ 10 \mapsto [0, 0.057] & 11 \mapsto [0.322, 0.674] \end{matrix} \right\}.$$

This yields a more precise output compared to Equation 6.13. If we check the results of the VQC, the probability associated with measuring q_0 equals 0 is $[0.176, 0.444]$ while we have q_0 equal to 1 with a probability in $[0.461, 1.04]$. Refining the abstraction confirms that the VQC in Figure 6.2 is robust when the initial input $x = [6, 2.7]$ is perturbed by an $\epsilon = 0.5$.

Why not a 'Symbolic Encoding'? This raises the question of whether performing a symbolic execution of the encoding part of the circuit could lead to improved precision. As we will show below, unfortunately, the answer is no.

Proposition 6.10. *The symbolic execution of the encoding part of a VQC does not improve precision.*

Proof. To prove the proposition, we show a counterexample. Let us consider the program $s := \mathtt{Rx}^q[x_0]; \mathtt{Ry}^q[x_1]$. We recall that the $\mathtt{Ry}$ is defined as: $\mathtt{Ry}[\theta] = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}$ and the $\mathtt{Rx}$ definition is given in Equation 6.1. The small program s models a circuit in which we are encoding multiple classical data in the same qubit. Let us consider $x = [\pi/2 - \epsilon, \pi/2 + \epsilon]$ and $y = [\pi/3 - \epsilon, \pi/3 + \epsilon]$, where $\epsilon = 1$. First, we can execute the circuit naively, starting

from the abstract state $\overline{\mathtt{V}}_0 = \{0 \mapsto \langle [1,1][0,0]\rangle, 1 \mapsto \langle [0,0],[0,0]\rangle\}$. In particular,

$$\overline{\mathtt{V}}_1 = [\![\mathtt{Ry}[\mathtt{y}]^q]\!]^\sharp_{\mathtt{S}}[\![\mathtt{Rx}^q[x]]\!]^\sharp_{\mathtt{S}}(\overline{\mathtt{V}}_0) = \left\{ \begin{array}{c} 0 \mapsto \langle [0.564, 0.66] + [0.306, 0.402]\rangle \\ 1 \mapsto \langle [0.306, 0.402] + [-0.66, -0.564]\rangle \end{array} \right\}.$$

We now execute the circuit keeping x and y symbolic. First we compute $[\![\mathtt{Rx}^q[x]]\!]^\sharp_{\mathtt{S}}(\overline{\mathtt{V}}_0)$, that results in the symbolic state:

$$\overline{\mathtt{T}}_1[x] = \left\{ \begin{array}{c} 0 \mapsto \cos(x/2) \\ 1 \mapsto -\iota \cdot \sin(x/2) \end{array} \right\}.$$

Then we compute symbolically:

$$[\![\mathtt{Ry}[\mathtt{y}]^q]\!]^\sharp_{\mathtt{S}}(\overline{\mathtt{T}}_1[x]) = \begin{bmatrix} \cos(y/2) \cdot \cos(x/2) + (-\sin(y/2)) \cdot (-\iota \sin(x/2)) \\ \sin(y/2) \cdot \cos(x/2) + \cos(y/2) \cdot (-\iota \sin(x/2)) \end{bmatrix}.$$

By substituting the values of x and y into the resulting state, we obtain $\overline{\mathtt{V}}_1$. Alternatively, we can use trigonometric formulas to manipulate the state, obtaining:

$$\overline{\mathtt{T}}_2[x,y] = \left\{ \begin{array}{l} 0 \mapsto \langle \left(\cos\left(\frac{x+y}{2}\right) + \cos\left(\frac{x-y}{2}\right)\right)/2],\ \left(\cos\left(\frac{x-y}{2}\right) - \cos\left(\frac{x+y}{2}\right)\right)/2\rangle \\ 1 \mapsto \langle \left(\sin\left(\frac{x+y}{2}\right) - \sin\left(\frac{x-y}{2}\right)\right)/2, -\left(\sin\left(\frac{x-y}{2}\right) - \sin\left(\frac{x+y}{2}\right)\right)/2\rangle \end{array} \right\}.$$

Now substituting the x and the y with the proper intervals, we have:

$$\overline{\mathtt{T}} = \left\{ \begin{array}{c} 0 \mapsto \langle [0.548,0.670][0.291,0.413]\rangle, \\ 1 \mapsto \langle [0.290, 0.413][-0.670, -0.548]\rangle \end{array} \right\},$$

that is noisier than $\overline{\mathtt{V}}_1$. □

We have shown, with a simple example, that the symbolic encoding does not always improve accuracy. This is because trigonometric substitutions do not always improve precision.

6.7 Evaluation

In this section, we evaluate our abstract interpretation-based framework for the robustness verification of VQC classifiers. In particular, we start by considering two standard dataset benchmarks in the classification literature, namely Iris [Ped+11] and MNIST [LCB10]. To keep the thesis self-contained, we briefly summarize the main characteristics of these datasets here. The Iris dataset consists of 150 samples of iris flowers classified into three species, such as *Iris setosa*, *Iris virginica*, and *Iris versicolor*. The input to the classifier consists

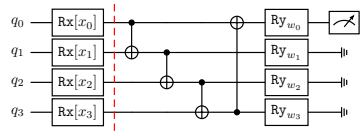


Figure 6.7: The QCL model, used to classify a 4-feature input data.

Figure 6.8: The CCQC model used on the iris dataset. Here $\texttt{R3}_{w_i,w_j,w_k} = \texttt{Rz}_{w_k} \cdot \texttt{Ry}_{w_j} \cdot \texttt{Rx}_{w_i}$

of four features, such as the length and width of the sepals and petals, all in centimeters, respectively. In our setting, we only consider a binary classification task between *Iris setosa* and *Iris versicolor*. The MNIST dataset of handwritten digits, containing 60,000 training and 10,000 testing examples, each represented as a grayscale image of size 28×28 pixels. In our setting, we consider the binary classification of downscaled to 4×4 image digits 0 and 1 (denoted `MNIST[0,1]`), and digits 2 and 6 (denoted `MNIST[2,6]`).

Although verification typically assumes a pre-trained model, in order to assess how different levels of accuracy affect the robustness of the VQC, we train and verify three distinct models using our proposed framework as VQC-based classifiers. Specifically:

- **QCL** [Mit+18] is a representative hybrid classical-quantum framework that employs a VQC composed of a nonlinear quantum encoding circuit for input data and a low-depth variational quantum circuit. We train and verify this model on Iris. The circuit is represented in Figure 6.7.
- **CCQC** [Sch+20] is a low-depth hybrid VQC framework designed for supervised learning. See Figure 6.8 for the circuit representation. Unlike the previous approach, classification is performed based on the measurement outcomes combined with a trainable bias term. We train and verify this model on the Iris dataset. The circuit is represented in Figure 6.8.
- The final models [HR24], which we refer to as **PV**, are a variation of the previous approaches and employ angle encoding [Mun24] and a parametric circuit to mitigate flat cost function landscapes during optimization [Gra+19]. We train this model on MNIST. The circuit is represented in Figure 6.9.

Training Phase. As described in Section 6.1, a VQC, like a deep NN, consists of a parameterized quantum circuit whose parameters are optimized during training to minimize a cost function, typically defined as the margin between the predicted and target classes [Bia+17]. For clarity, Algorithm 3 presents a simplified version of the training algorithm used for the QCL model on the Iris dataset, while in Figure 6.11, we provide an illustrative view of the

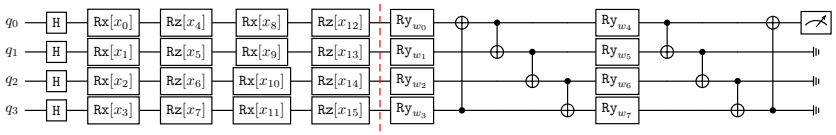


Figure 6.9: The PV model, used to classify a 16-feature input data.

decision boundaries for the CCQC model on the same dataset. Specifically, we adopt a supervised training procedure for each VQC tested, exploiting the Pennylane framework [Ber+18]. For each training example, the prediction is computed via a VQC, and the parameters θ are updated using gradient-based optimization. This process is repeated over multiple epochs to improve classification performance. While the pseudocode illustrates a generic Stochastic Gradient Descent (SGD) update, in our experiments, we employ different optimization strategies, such as PennyLane's *NesterovMomentumOptimizer* [Ber+18], using the quantum state encoding defined in [Mot+04]. The table in Figure 6.11 shows the test set accuracy for each model tested on its corresponding dataset.

Algorithm 3 Training using SGD for QCL Binary Classification

Input: QCL with parameter θ, Training data $\{(X_i, Y_i)\}_{i=1}^{N}$, learning rate η, epochs E
Initialize: Random parameters θ
for epoch $= 1$ to E **do**
  for each (X_i, Y_i) in training set **do**
    $\hat{y}_i \leftarrow \mathrm{QCL}(X_i, \theta)$
    $loss \leftarrow (\hat{y}_i - Y_i)^2$
    $g \leftarrow \nabla_\theta \ell_i$ ▷ Compute gradient
    $\theta \leftarrow \theta - \eta \cdot g$ ▷ Update parameters
  end for
  $a \leftarrow \textsc{Accuracy}(\text{training set}, QCL, \theta)$
end for
return trained parameters θ

Verification Phase. After the training phase, we perform a formal verification step to assess the robustness of trained VQCs to input perturbations. To this end, we extend the Pennylane framework [Ber+18] to support our abstract interpretation-based formulation, which operates over intervals of real and complex numbers. This extension enables reachability analysis through the circuit, allowing us to reason about how input uncertainty propagates through quantum operations.

The verification process is summarized on the left side in Algorithm 4. In detail, we show the algorithm used to compute the maximum perturbation ϵ each model can tolerate for a given input x and target class y. The Verify function is the core of the pipeline. It

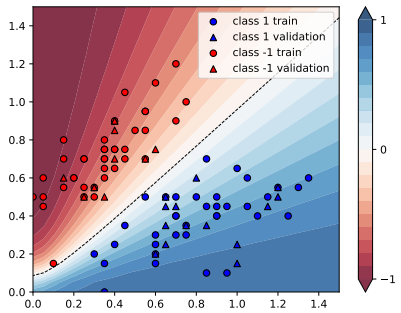


Figure 6.10: Empirical decision boundary on the Iris dataset using CCQC classifier (100% accuracy).

takes as input a trained VQC (i.e., with fixed parameters θ), a test input x, a class label y, and a perturbation radius ϵ. It encodes the input as intervals, i.e., each feature of x is extended to $[x_i - \epsilon, x_i + \epsilon]$, and propagates these intervals through the encoding and a symbolic representation of the variational layers of the circuit.

During interval propagation, both real and complex-valued amplitudes are tracked using interval arithmetic. The prediction is derived by identifying the output class whose associated probability interval has a lower bound that exceeds the upper bounds of all others, ensuring the prediction is unambiguous under the entire input region. If this condition is not met, for instance if the intervals of two or more output classes overlap, an iterative refinement process is employed: the input interval with the largest uncertainty (i.e., the largest width) is split, and the verification is recursively applied to the sub-regions until a conclusive result is reached (i.e., all the subregions are safe or a single counterexample is discovered) or a specified precision is met.[‡‡] Broadly speaking, the algorithm combines an exponential and binary search. It starts from a minimal ϵ_{min} and iteratively doubles it until the robustness condition is violated or ϵ_{max} is reached. Then, a binary search is used to converge to the largest ϵ that still guarantees robustness, with a desired precision τ.

In Figure 6.11, the verification results are reported on different quantum classifiers and datasets. The results reveal several interesting insights: `QCL-Iris` achieves an accuracy of 76% with a mean maximum ϵ of 0.07496. This relatively high perturbation tolerance reflects the low dimensionality and structured nature of the Iris dataset. On the other hand, `CCQC-Iris`, which adopts a different encoding or circuit structure, achieves perfect

[‡‡] We also experimented with different heuristics for selecting the node to split during refinement—for example, choosing nodes at random—but the strategy described above consistently yielded the best results in terms of precision and was therefore used to generate the reported outcomes.

Robustness Verification results

Model-Dataset	Accuracy	Mean of max ϵ pert.	Mean Time
QCL-Iris	76%	0.07496±0.05	128.2 s
CCQC-Iris	100%	0.1216±0.05	10.10 s
PV-MNIST[0,1]	95%	0.0048±0.002	570.35 s
PV-MNIST[2,3]	61%	0.0005±0.001	78s

Figure 6.11: Robustness verification results. The mean of the maximum ϵ is computed over 10 randomly selected inputs from the test set and the model with the highest accuracy during training.

accuracy (100%) and significantly higher robustness (mean $\epsilon = 0.1244$). This indicates better alignment between circuit expressivity and the problem structure, possibly due to improved use of parameterized gates. For the more complex MNIST dataset, particularly in binary classification tasks over digits [0,1] and [2,6], we observe a clear drop in robustness: 0.0048 and 0.0022, respectively. While `PV-MNIST[0,1]` still maintains high accuracy (95%), the low maximum ϵ highlights how small input perturbations can significantly alter the classification outcome in high-dimensional settings. This is even more evident for `PV-MNIST[2,6]`, where both accuracy and robustness are lower, suggesting that distinguishing these digits is harder for the model.

Algorithm 4 Compute max ϵ perturbation

Input: VQC with trained parameter θ, input x, y class, $\epsilon_{\min}$, $\epsilon_{\max}$, τ tolerance threshold
$\epsilon \leftarrow \epsilon_{\min}$
while $\epsilon \leq \epsilon_{\max}$ **and** VERIFY(VQC, x, ϵ, y) == robust **do**
    $\epsilon \leftarrow 2 \cdot \epsilon$
end while
$high \leftarrow \min(\epsilon, \epsilon_{\max})$
$low \leftarrow \epsilon/2$
$max\_epsilon \leftarrow low$
while $high - low > \tau$ **do**
    $mid \leftarrow (low + high)/2$
    if VERIFY(VQC, x, ϵ, y) == robust **then**
        $max\_epsilon \leftarrow mid$
        $low \leftarrow mid$
    else
        $high \leftarrow mid$
    end if
end while
return $max\_epsilon$

Overall, the results validate the effectiveness of our reachability-based verification approach in quantifying model robustness in quantum machine learning. The method not only provides

formal guarantees but also reveals the limitations of current quantum models in handling high-dimensional, less separable data. These insights could drive the design of more robust circuits and encoding schemes in future work.

6.8 Related Work

In this section, we recall the related work that is not mentioned in the survey chapter (Chapter 3).

NN verification. Classical NN formal verification tools [Liu+21; Wei+25], such as those developed in [Geh+18] and [Sin+19], adopt abstract interpretation techniques to conservatively approximate the network's behavior by propagating abstract domains (like intervals or zonotopes) through its layers. As demonstrated in this work, this approach allows efficient and sound verification of properties—such as robustness—by computing conservative bounds on the output set. A different category of tools, including CROWN, α-CROWN, and β-CROWN [Zha+18; Xu+20; Wan+21], employs linear relaxation methods to approximate nonlinear activations with tight linear bounds, offering a balance between scalability and precision. These techniques are often integrated with Branch-and-Bound frameworks [Bun+18], such as MN-BaB [Fer+22], which partition the verification task into smaller subproblems—either by splitting the input perturbation space [Wan+18] or by dividing ReLU activations into linear segments [Bun+18; Bun+20]. This combination enhances completeness and precision, though at the cost of increased computational complexity. More recently, emerging approaches have extended the scope of verification to identify all input regions that satisfy a given output specification [DGM19; Mar+23; Mar+24; Kot+23; ZWK24].

VQC Robustness Evaluation. Robustness evaluation of variational quantum circuits remains largely unexplored. However, recent works have started to address this challenge. For example, QuanTest [Shi+25] proposes an adversarial testing framework that generates inputs maximizing a quantum entanglement adequacy criterion while exposing erroneous behaviors, using a gradient-based joint optimization approach. Other studies [LDD20; WTD24] have investigated the vulnerability of VQC models to adversarial attacks, demonstrating that quantum classifiers, like their classical counterparts, can be misled by imperceptible perturbations on both classical and quantum inputs across a variety of tasks.

Robustness guarantees have been studied in prior work. Weber et al. [Web+21] establish a connection between quantum hypothesis testing and robustness, deriving tight necessary and sufficient conditions for noise tolerance. Guan et al. [GFY21] propose a fidelity-based

framework with robustness bounds computable via semidefinite programming (SDP). However, both approaches model noise as acting on quantum states, i.e., after the encoding stage, and thus cannot capture perturbations affecting the classical input data. In contrast, our framework explicitly models noise injected at the input level of variational quantum circuits (VQCs), allowing us to analyze how noise propagates through the encoding—something not addressed by previous methods.

6.9 Discussion and Future Directions

In this chapter, we have presented a novel framework for the formal verification of variational quantum circuits, which is based on the abstract interpretation theory. Our approach is grounded in interval abstractions, which, despite their non-relational nature and the resulting overapproximations, still enable effective verification, as demonstrated by our empirical evaluation. To enhance precision, we have introduced several strategies inspired by techniques developed for classical deep neural network verification.

We have observed that classification based on the abstract execution of a VQC may suffer from overestimation errors, preventing the system from producing a concrete output. However, recent work on classical NNs [GMP24; MMF25] has shown that it is possible to relax the requirement of obtaining a precise answer by instead providing an abstract robustness guarantee. This involves applying abstraction not only to the input but also to the output, allowing for different degrees of precision. The verification task then becomes a question of whether the desired property holds within a specified level of abstraction, an idea closely related to Adequacy in static analysis [Mas24]. We argue that this tunable approach to verification is promising and deserves further investigation.

Another promising direction for future work is to explore alternative techniques—beyond those discussed in Section 6.6—for recovering precision. In particular, methods such as circuit rewriting and simplification, including the use of ZX-calculus [CD09; Wet20], offer a compelling avenue. These techniques could be applied as a pre-processing step to simplify quantum circuits, for instance, by reducing the number of phase gates in the encoding, thereby potentially enhancing the overall precision of the analysis. The key challenge lies in understanding how to effectively integrate this powerful formalism into our verification framework and adapt it to our specific context.

Alternative representations of quantum states also offer valuable opportunities. One possibility could be the path-sum formalisms [Amy19; AL25; Ric65; Amy18; Cha+21], which provide symbolic descriptions of quantum circuits that can be more concise than vector-based representations, and have been shown to be effective for verifying large Clifford+T circuits.

Similarly, bit-wise simulation techniques [CS23; DD21] allow memory-efficient encoding of quantum states, especially when entanglement is limited. We also plan to explore the use of the Ellipsoid domain [Fer04; Cou+06] to capture second-order relations between the amplitudes of a quantum state.

Finally, we plan to extend our framework to support the analysis of hybrid quantum-classical architectures [Hav+19; Mat+21; Mat+22], in which VQCs appear as components within larger machine learning pipelines. This includes exploring how input abstractions can propagate through both quantum and classical layers, as well as how robustness guarantees can be preserved end-to-end.

Chapter 7

A Denotational Semantics for Quantum Loops

A crucial part of any computer program is its control flow. In classical computing, control flow determines how instructions are sequenced and branched, allowing programs to make decisions and change behavior based on specific conditions. This naturally includes the ability to express loops controlled by conditional guards. In the quantum setting, however, control flow cannot be interpreted in the same way, since quantum programs operate on devices governed by the laws of quantum mechanics. A quantum processor acts on *superpositions* of states (qubits) rather than on single deterministic states, and can generate *entangled* states, non-classical correlations that link multiple qubits together. As we will discuss in Section 7.7, most quantum languages feature quantum data combined with classical probabilistic control based on measurement operations. While this approach is convenient for implementation, it prevents the representation of programs that evolve in a superposition of different execution paths.

In this chapter, we aim to describe a quantum program that is as general as possible by avoiding an explicit syntactic construct for measurement, which would effectively lead to a (classical) probabilistic semantics of the program. At this point, a crucial question arises: *how can we handle termination in quantum loops?* In the classical Turing model, a *halting bit* can signal the end of computation. However, as shown in [LP98; MO05; Mye97; Shi02; Son08], defining a similar *halting qubit* in the quantum setting is impossible without compromising the computation itself. In contrast to classical loops, measuring a quantum guard collapses the superposition into a classical state, thereby altering the

computation. Quantum languages that rely on measurement-based control flow address this issue by sacrificing unitarity. Since genuine quantum computation proceeds through unitary transformations, determining externally whether a quantum execution has reached a fixed point would require either stopping the computation after a finite number of steps or letting it evolve indefinitely. Restricting to finite computations might seem reasonable, but it prevents the definition of a meaningful semantics for reasoning about termination. Therefore, as in the classical case, it is essential to have a semantics capable of capturing both terminating and non-terminating computations, though achieving this in the quantum setting is more challenging due to the issues outlined above.

We show that it is possible to model which parts of a quantum loop terminate and which do not by approximating the concrete unitary behavior of a quantum program through a sequence of linear operators. These operators effectively *separate* the subspaces corresponding to terminated executions from those still evolving, even in the limit of an infinite loop. By relaxing the unitarity constraint, we can define an approximating sequence of linear operators that converges to a well-defined limit. For finite computations, this limit coincides with the true (unitary) behavior of the quantum program; in the case of infinite computations, it is a linear operator with norm strictly less than one, since the operator does not represent the portion of the superposition that is still being computed. This approximating denotational semantics is expressive enough to capture precisely the program's observable input-output behavior, which we demonstrate by establishing a correspondence between the denotational semantics, defined through bounded linear operators, and the unitary operators semantics.

7.1 A Quantum while language

To present our semantics, we introduce a generic quantum language with a minimal set of constructs consisting of quantum loop iteration, sequential composition, and unitary transformation. The syntax of our simple language, which we will refer to as SL, is given by the following grammar defining a program s as follows:

$$s ::= U(\mathbf{q}) \mid s; s \mid \texttt{skip} \mid \texttt{qwhile}\ q\ \texttt{do}\ \{s\}, \tag{7.1}$$

where q is a quantum variable and $\mathbf{q}$ denotes a sequence $q^1, \ldots, q^n$ of quantum variables. In SL, we do not define a command for the state initialization (or assignment) since we assume that all variables are initialized to $|0 \ldots 0\rangle$. All operations on variables are performed by the statement $U(\mathbf{q})$ corresponding to applying a unitary transformation U to $\mathbf{q}$. Every possible quantum transformation, except measurement, is a composition of unitary transformations. The principle of deferred measurement [NC10, Chapter 4] guarantees

that all intermediate measurements in a quantum circuit can be moved to the end of the computation; thus, excluding measurement operations from our language does not cause a loss of generality. Since measurement does not occur in SL programs, the semantics of such programs is a deterministic transition between quantum states according to the physical law (the Shrödinger equation) governing the quantum system on which the program is intended to be executed, without having to establish when quantum mechanics should leave space to classical mechanics, which essentially constitutes the measurement problem in the interpretation of quantum theory.

7.1.1 The State Space

To define a semantics for SL, we consider a domain similar to that described in [Yin16, Chapter 3], namely an infinite-dimensional Hilbert space obtained by composing, via tensor product, the vector spaces associated with each program variable.

Given a program s, let Q_s be the set of all variables occurring in s. Each quantum variable $q \in Q_s$ has a type $\mathscr{H}_q$, i.e. its values are vectors $|\varphi\rangle_q$ in the Hilbert space $\mathscr{H}_q$ (or quantum state space of q). We call a single qubit a Boolean variable, as its type is the 2-dimensional Hilbert space $\mathscr{H} = \{\alpha |0\rangle + \beta |1\rangle \mid \alpha, \beta \in \mathbb{C}\}$. In general, for any $n \in \mathbb{N}$, the type (or state space) of an n-qubit variable is the 2^n-dimensional Hilbert space $\mathscr{H}^{\otimes n}$, whose vectors are uniquely determined by their coordinates with respect to a basis of 2^n orthonormal vectors. We define the state space of a quantum program s in the SL language as the composition of the state spaces associated with each variable in Q_s:

$$\mathscr{H}_{Q_s} = \bigotimes_{q \in Q_s} \mathscr{H}_q.$$

In order to deal with (possibly infinite) loops, we will need to consider (as explained later) a countable infinite set $T = \{t_i\}$ of auxiliary quantum Boolean variables and their composed Hilbert space:

$$\mathscr{H}_T = \bigotimes_{t_i \in T} \mathscr{H}_{t_i}.$$

We then define the Hilbert space of a program s as $\mathscr{H}_P = \mathscr{H}_T \otimes \mathscr{H}_{Q_s}$.

7.2 Unitary Semantics

A quantum language with no classical operations can be completely described by using unitary operators that can be visually represented by quantum circuits. A mathematical description of these circuits is by the group of unitary operators on a Hilbert space.

Figure 7.1: Quantum circuits corresponding to **(a)** $U(\mathbf{q})$ and **(b)** $s_1; s_2$.

Firstly, we can define the semantics for each statement in SL except for the `while` statement.

Definition 7.1. *Let $\mathcal{U}(\mathscr{H}_P)$ be the group of unitary operators from $\mathscr{H}_P$ to $\mathscr{H}_P$. The unitary semantics is the function $[\![\cdot]\!] : s \to \mathcal{U}(\mathscr{H}_P)$, defined by:*

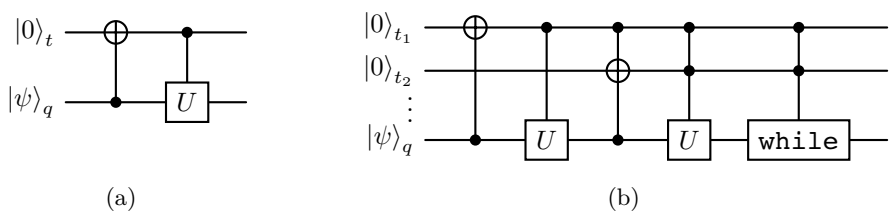
(1) $[\![\texttt{skip}]\!] = \mathbf{I}_{\mathscr{H}_P}$;
(2) $[\![U(\mathbf{q})]\!] = \mathbf{U}$, *where* $\mathbf{U} = \mathbf{I} \otimes U \otimes \mathbf{I}$, *i.e. the extension of* U *on* $\mathscr{H}_P$;
(3) $[\![s_1; s_2]\!] = [\![s_2]\!] \cdot [\![s_1]\!]$,

Rule (**2**) corresponds to computing the operator U on $\mathbf{q}$, and the identity on the other components of the program space. Rule (**3**) models sequential composition by means of matrix multiplication (the group operation). Since $\mathbf{U}$ and $\mathbf{I}$ are unitary operators, the semantics of each statement is a unitary operator. Therefore, every statement can be represented by a circuit as shown in Figure 7.1. The case for the while is slightly more involved, and we treat it in the next section.

7.2.1 While Loop Semantics

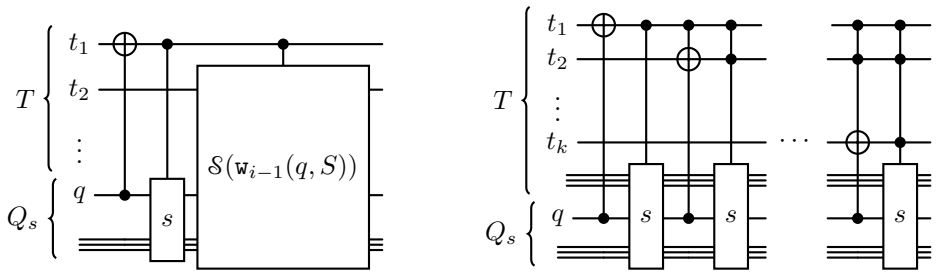
A well-formed `while` statement should modify its guard within the loop body. To define a quantum `while` instruction, we need a representation where the guard qubit can be updated while preserving the unitarity of the evolution. Ideally, given a program $p = \texttt{qif}\ q\ \texttt{do}\ \{U(q)\}$ *, we would like to have a unitary operator C such that $C(\alpha\,|0\rangle_q + \beta\,|1\rangle_q) = \alpha\,|0\rangle_q + \beta U\,|1\rangle_q$. However, in quantum computing, controlled operations cannot have their controller affected by the target, as this would break unitarity. For example, consider $p = \texttt{qif}\ q\ \texttt{do}\ \{X(q)\}$. The corresponding operator C would map $|0\rangle_q$ to $|0\rangle_q$ and $|1\rangle_q$ to $|0\rangle_q$, which is clearly a non-injective, and thus non-unitary map. A way to overcome this problem is to model a self-controlled operation by introducing an auxiliary qubit on which to copy information via a `CX` gate. Figure 7.2a illustrates a hypothetical quantum if-statement where the guard is included in the body. The semantics can be represented by a unitary operator CU such that $CU(|0\rangle_t \otimes (\alpha\,|0\rangle_q + \beta\,|1\rangle_q)) = \alpha\,|0\rangle_t\,|0\rangle_q + \beta\,|1\rangle_t\,U\,|1\rangle_q$. Extending this approach to a

* We use here the '`qif do {}`' notation as a shortcut for indicating one iteration of the `qwhile do {}` statement.



(a) (b)

Figure 7.2: Quantum circuits for **(a)** `qif` q `do` $\{U(q)\}$ and **(b)** `qwhile` q `do` $\{U(q)\}$



(a) (b)

Figure 7.3: A circuit representation of the operator defined by Equation 7.2 (a) and of a quantum while loop bounded to k iterations

while loop introduces an additional challenge: each iteration requires a fresh temporary qubit, which implies the need for an infinite set of auxiliary qubits to model non-terminating loops. Figure 7.2b shows the circuit representation of a quantum while loop, which is recursively defined and corresponds to an infinite composition of unitary operations on an infinite-dimensional Hilbert space.

To define a semantics for the while loop construct, it will be convenient to look at the program state space as the Hilbert space $\mathscr{H}_{s'} = \mathscr{H}_g \otimes \mathscr{H}_s$, where we separate the 2-dimensional Hilbert space, $\mathscr{H}_g$, representing the control (guard) qubit, and the space, $\mathscr{H}_s$, of target qubits. Given a unitary operator U_s on $\mathscr{H}_s$, the unitary operator corresponding to U_s controlled by g can be represented by $|0\rangle\langle 0|_g \otimes \mathbf{I}_s + |1\rangle\langle 1|_g \otimes U_s$, where $\mathbf{I}_s$ is the identity on $\mathscr{H}_s$ and $|0\rangle\langle 0|_g$ and $|1\rangle\langle 1|_g$ are the projectors on the basis vectors $|0\rangle_g$ and $|1\rangle_g$ of $\mathscr{H}_g$, respectively.

Proposition 7.1. *Let* $\mathtt{P}_{i_g} = (|i\rangle\langle i|_g \otimes \mathbf{I}_s)$ *be the projector* $|i\rangle\langle i|_g$ *extended to the whole* $\mathscr{H}$*, and let* $\mathbf{U} = (\mathbf{I}_g \otimes U_s)$ *be the extension of* U_s *in* $\mathscr{H}_{s'}$*. The controlled operation* $|0\rangle\langle 0|_g \otimes \mathbf{I}_s + |1\rangle\langle 1|_g \otimes U_s$ *is equivalent to the operator* $\mathtt{P}_{0_g} + \mathtt{P}_{1_g} \cdot \mathbf{U}$*.*

Proof. By the property $AC \otimes BD = (A \otimes B)(C \otimes D)$ [RC91, Lemma 4.2.10] we have $|1\rangle\langle 1|_g \otimes U_s = (|1\rangle\langle 1|_g \cdot \mathbf{I}_g) \otimes (\mathbf{I}_s \cdot U_s) = (|1\rangle\langle 1|_g \otimes \mathbf{I}_s)(\mathbf{I}_g \otimes U_s) = \mathtt{P}_{1_g} \cdot \mathbf{U}$. Thus $|0\rangle\langle 0|_g \otimes \mathbf{I}_s + |1\rangle\langle 1|_g \otimes U_s = \mathtt{P}_{0_g} + \mathtt{P}_{1_g} \cdot \mathbf{U}$. □

As shown in Figure 7.2b, to perform a quantum loop, we need to make a quantum copy of the guard variable in a new, fresh ancillary variable for every iteration. This is realized by a CX gate, which we will denote by $\mathbf{G}(q,n)$ in our semantics, where q is a variable in Q_s and $n \in \mathbb{N}$, $n \geq 1$ indicates the target $t_n \in T$.

Given a qubit $q \in Q_s$, and a unitary operator S that does not act on t_1, we recursively define an operator $\mathtt{w}_n(q,S)$, using the control operation in the form introduced in Proposition 7.1, as follows:

$$\begin{aligned} \mathtt{w}_0(q,S) &= \mathbf{I} \\ \mathtt{w}_n(q,S) &= (\mathtt{P}_{0_{t_1}} + \mathtt{P}_{1_{t_1}} \cdot \mathcal{S}(\mathtt{w}_{n-1}(q,S)) \cdot S)\mathbf{G}(q,1), \end{aligned} \tag{7.2}$$

where $\mathcal{S}$ shifts the controls to keep the first ancilla qubit free, i.e., $\mathcal{S}(\mathbf{G}(q,n)) = \mathbf{G}(q,n+1)$ and $\mathcal{S}(\mathtt{P}_{j_{t_n}}) = \mathtt{P}_{j_{t_{n+1}}}$. Concretely, the operator $\mathtt{w}_i(q,S)$ corresponds to the circuit depicted in Figure 7.3a, where the component $\mathbf{G}(q,1)$ is represented by the first controlled-not operation, and the expression $(\mathtt{P}_{0_{t_1}} + \mathtt{P}_{1_{t_1}} \cdot \mathcal{S}(\mathtt{w}_{n-1}(q,S))S)$ corresponds to the operation S followed by $\mathcal{S}(\mathtt{w}_{n-1}(q,S))$ controlled by t_1. Here, $\mathcal{S}(\mathtt{w}_{n-1}(q,S))S$ denotes the composition of the unitary S, which encodes the semantics of the body of the while loop, and $\mathcal{S}(\mathtt{w}_{n-1}(q,S))$, which captures the semantics of the remaining part of the while loop.

Proposition 7.2. *The closed formula of Equation 7.2, is:*

$$\mathtt{W}_n(q,S) = \sum_{k=1}^{n} \left(\prod_{i=1}^{k-1} (\mathtt{P}_{1_{t_i}}) \cdot \mathtt{P}_{0_{t_k}} \cdot \prod_{i=n-k}^{n-2} \big(\boldsymbol{G}(q,n-i) \cdot S\big) \cdot \boldsymbol{G}(q,1) \right) + \tag{7.3}$$

$$+ \prod_{i=1}^{n} \mathtt{P}_{1_{t_i}} \cdot \prod_{i=0}^{n-1} \big(S \cdot \boldsymbol{G}(q,n-i)\big). \tag{7.4}$$

Proof. We proceed by induction. If $n = 0$, $\mathtt{w}_0 = \mathbf{I}$ and

$$\mathtt{W}_0 = \sum_{k=1}^{0} (\dots) + \prod_{i=1}^{0} \mathtt{P}_{1_{t_i}} \cdot \prod_{i=0}^{-1} [\,s\,]\mathbf{G}(q,n-i) = \mathbf{I}.$$

Let's consider $\mathtt{w}_{n+1} = (\mathtt{P}_{0_{t_1}} + \mathtt{P}_{1_{t_1}} \cdot \mathcal{S}(\mathtt{W}_n) \cdot [\,s\,])\mathbf{G}(q,1)$. By inductive hypothesis $\mathtt{w}_n = \mathtt{W}_n$, thus we need to compute $\mathcal{S}(\mathtt{W}_n)$:

$$\begin{aligned} \mathcal{S}(\mathtt{W}_n) = &\sum_{k=1}^{n} \left(\prod_{i=1}^{k-1} (\mathtt{P}_{1_{t_{i+1}}}) \cdot \mathtt{P}_{0_{t_{k+1}}} \cdot \prod_{i=n-k}^{n-2} (\mathbf{G}(q,n-i+1)[\,s\,]) \cdot \mathbf{G}(q,2) \right) + \\ &+ \prod_{i=1}^{n} \mathtt{P}_{1_{t_{i+1}}} \cdot \prod_{i=0}^{n-1} [\,s\,]\mathbf{G}(q,n-i+1). \end{aligned}$$

The initial and final values of the first and third products can be updated to remove the $+1$

from the indices, resulting in:

$$\mathcal{S}(\mathtt{W}_n) = \sum_{k=1}^{n} \left(\prod_{i=2}^{(k+1)-1} (\mathtt{P}_{1_{t_i}}) \cdot \mathtt{P}_{0_{t_{k+1}}} \cdot \prod_{i=n-k}^{n-2} (\mathbf{G}(q,(n+1)-i)[\,s\,]) \cdot \mathbf{G}(q,2)) \right) + \\ + \prod_{i=2}^{n+1} \mathtt{P}_{1_{t_i}} \cdot \prod_{i=0}^{n-1} [\,s\,]\mathbf{G}(q,(n+1)-i).$$

Finally, the $k+1$ can be collected, and the summation indices can be updated, resulting in:

$$\mathcal{S}(\mathtt{W}_n) = \sum_{k=2}^{n+1} (\prod_{i=2}^{k-1} \left(\mathtt{P}_{1_{t_i}}) \cdot \mathtt{P}_{0_{t_k}} \cdot \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q,(n+1)-i)[\,s\,]) \cdot \mathbf{G}(q,2)) \right) + \\ + \prod_{i=2}^{n+1} \mathtt{P}_{1_{t_i}} \cdot \prod_{i=0}^{n-1} [\,s\,]\mathbf{G}(q,(n+1)-i).$$

Now we substitute this equation in $\mathtt{w}_{n+1} = \mathtt{P}_{0_{t_1}} \cdot \mathbf{G}(q,1) + \mathtt{P}_{1_{t_1}} \cdot \mathcal{S}(\mathtt{W}_n) \cdot [\,s\,] \cdot \mathbf{G}(q,1)$. Initially, we compute $\mathtt{w}' = \mathtt{P}_{1_{t_1}} \cdot \mathcal{S}(\mathtt{W}_n)$, in particular:

$$\begin{aligned}
\mathtt{w}' =& \mathtt{P}_{1_{t_1}} \big(\sum_{k=2}^{n+1} (\prod_{i=2}^{k-1} (\mathtt{P}_{1_{t_i}}) \mathtt{P}_{0_{t_k}} \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q,(n+1)-i)[\,s\,]) \mathbf{G}(q,2))) + \\
& + \prod_{i=2}^{n+1} \mathtt{P}_{1_{t_i}} \prod_{i=0}^{n-1} [\,s\,]\mathbf{G}(q,(n+1)-i) \big) = \\
=& \sum_{k=2}^{n+1} (\mathtt{P}_{1_{t_1}} \prod_{i=2}^{k-1} (\mathtt{P}_{1_{t_i}}) \mathtt{P}_{0_{t_k}} \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q,(n+1)-i)[\,s\,]) \mathbf{G}(q,2)) + \\
& + \mathtt{P}_{1_{t_1}} \prod_{i=2}^{n+1} \mathtt{P}_{1_{t_i}} \prod_{i=0}^{n-1} [\,s\,]\mathbf{G}(q,(n+1)-i) = \\
=& \sum_{k=2}^{n+1} (\prod_{i=1}^{k-1} (\mathtt{P}_{1_{t_i}}) \mathtt{P}_{0_{t_k}} \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q,(n+1)-i)[\,s\,]) \mathbf{G}(q,2)) + \\
& + \prod_{i=1}^{n+1} \mathtt{P}_{1_{t_i}} \prod_{i=0}^{n-1} [\,s\,]\mathbf{G}(q,(n+1)-i).
\end{aligned}$$

Secondly, let's consider $\mathtt{w}'' = \mathtt{w}' \cdot [\![s]\!] \cdot \mathbf{G}(q,1)$, it result in:

$$\begin{aligned}
\mathtt{w}'' =& \Big(\sum_{k=2}^{n+1} (\prod_{i=1}^{k-1} (\mathtt{P}_{1_{t_i}}) \mathtt{P}_{0_{t_k}} \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q,(n+1)-i)[\![s]\!]) \mathbf{G}(q,2)) + \\
&+ \prod_{i=1}^{n+1} \mathtt{P}_{1_{t_i}} \prod_{i=0}^{n-1} [\![s]\!] \mathbf{G}(q,(n+1)-i) \Big) [\![s]\!] \mathbf{G}(q,1) = \\
=& \sum_{k=2}^{n+1} (\prod_{i=1}^{k-1} (\mathtt{P}_{1_{t_i}}) \mathtt{P}_{0_{t_k}} \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q,(n+1)-i)[\![s]\!]) \mathbf{G}(q,2) [\![s]\!] \mathbf{G}(q,1)) + \\
&+ \prod_{i=1}^{n+1} \mathtt{P}_{1_{t_i}} \prod_{i=0}^{n-1} [\![s]\!] \mathbf{G}(q,(n+1)-i) [\![s]\!] \mathbf{G}(q,1)
\end{aligned}$$

If we consider the second and fourth products, we can include $\mathbf{G}(q,2)[\![s]\!]$ by updating the indices. Specifically, in the first product, when $i = n-2$, we have $n+1-n+2 = 3$. Therefore, by setting $n-1$ as the upper limit of the product, we can include $\mathbf{G}(q,2)[\![s]\!]$ in the product. Similarly, in the fourth product, when $i = n-1$, $[\![s]\!]\mathbf{G}(q,2)$ is included. By varying the product from 0 to n, we also include $[\![s]\!]\mathbf{G}(q,1)$. So finally, we can write $\mathtt{w}''$ as:

$$\begin{aligned}
\mathtt{w}'' =& \sum_{k=2}^{n+1} \left(\prod_{i=1}^{k-1} (\mathtt{P}_{1_{t_i}}) \mathtt{P}_{0_{t_k}} \prod_{i=n-(k-1)}^{n-1} (\mathbf{G}(q,(n+1)-i)[\![s]\!]) \mathbf{G}(q,1) \right) + \\
&+ \prod_{i=1}^{n+1} \mathtt{P}_{1_{t_i}} \prod_{i=0}^{n} [\![s]\!] \mathbf{G}(q,(n+1)-i).
\end{aligned}$$

Finally, since $\mathtt{W}_{n+1} = \mathtt{P}_{0_{t_1}} \mathbf{G}(q,1) + \mathtt{W}''$ and $(\prod_{i=1}^{k-1} (\mathtt{P}_{1_{t_i}}) \mathtt{P}_{0_{t_k}} \prod_{i=n-(k-1)}^{n-1} (\mathbf{G}(q,(n+1)-i)[\![s]\!]) \mathbf{G}(q,1))_{k=1} = \mathtt{P}_{0_{t_1}} \cdot \mathbf{G}(q,1)$ we can write:

$$\begin{aligned}
\mathtt{W}_{n+1} =& \sum_{k=1}^{n+1} \left(\prod_{i=1}^{k-1} (\mathtt{P}_{1_{t_i}}) \mathtt{P}_{0_{t_k}} \prod_{i=(n+1)-k}^{n-1} (\mathbf{G}(q,(n+1)-i)[\![s]\!]) \mathbf{G}(q,1) \right) + \\
&+ \prod_{i=1}^{n+1} \mathtt{P}_{1_{t_i}} \prod_{i=0}^{n} [\![s]\!] \mathbf{G}(q,(n+1)-i),
\end{aligned}$$

i.e., we write $\mathtt{W}_{n+1}$ in the form of Equation 7.3.

□

Given a `qwhile` q `do` $\{s\}$ statement, we can then build a chain of its finite unitary approximation $\{\mathtt{W}(q,[\![s]\!]\}_n$, for all $n \in \mathbb{N}$. Unfortunately, Theorem 7.1 tells us that there is no limit to this sequence, and therefore our unitary semantics is not able to model infinite loops.

From [Van], we recall the notion of *strong convergence* and an important theorem about the convergence of an infinite sequence of operators.

Definition 7.2. *Let T_n and T be linear operators from $\mathscr{H}_P$ to itself. If $\|T_n|\psi\rangle - T|\psi\rangle\| \to 0$ as $n \to \infty$, $\forall |\psi\rangle \in \mathscr{H}_P$, then the sequence of operators $\{T_n\}$ is strongly convergent to T (denoted as $T_n \to T$).*

The following well-known result in the theory of normed linear spaces is stated for the general case of Banach spaces. Since Hilbert spaces are a special case of Banach spaces, the theorem obviously applies in our setting.

Theorem 7.1. *Let $\{T_n\}_n$ be a sequence of bounded linear operators from $X \to Y$, where X and Y are Banach spaces. $T_n \to T$ if and only if the sequence $\{\|T_n\|\}_n$ is bounded and the sequence $\{T_n x\}_n$ is a Cauchy sequence in Y for all $x \in M \subset X$, where the span of M is dense in X.*

We recall that a sequence $\{x_n\}_n$ in a Hilbert space is said to be a Cauchy sequence if, for any positive real number $\epsilon > 0$, there exists a positive integer N such that for all positive integers m and n greater than N, the distance $\|x_m - x_n\| < \epsilon$.

We can show that the sequence generated by unfolding the program `qwhile` q `do` $\{$`skip`$\}$, evaluated on $|\psi\rangle_P = |0\ldots\rangle_T |1\rangle_q$, is not Cauchy. In particular, we obtain the chain:

$$\begin{aligned} \mathtt{W}_0(q, X_q)\,|0\ldots\rangle_T\,|1\rangle_q &= |0\ldots\rangle_T\,|1\rangle_q\,, \\ \mathtt{W}_1(q, X_q)\,|0\ldots\rangle_T\,|1\rangle_q &= |10\ldots\rangle_T\,|1\rangle_q\,, \\ \mathtt{W}_n(q, X_q)\,|0\ldots\rangle_T\,|1\rangle_q &= \left|1^{\otimes n}0\ldots\right\rangle_T |1\rangle_q\,. \end{aligned} \tag{7.5}$$

We see that for all $n \in \mathbb{N}$, $\|\mathtt{W}_{n+1}(q, X_q)\,|\psi\rangle_P - \mathtt{W}_n(q, X_q)\,|\psi\rangle_P\| = 2$. More in general, if

$$|\psi\rangle_P = |0\ldots\rangle_T\,(\alpha\,|0\rangle + \beta\,|1\rangle)_q,$$

we have

$$\mathtt{W}_n(q, X_q)\,|\psi\rangle_P = \alpha\,|0\ldots\rangle_T\,|0\rangle_q + \beta\left|1^{\otimes n}0\ldots\right\rangle_T |1\rangle_q\,,$$

and for all $n \in \mathbb{N}$, $\|\mathtt{W}_{n+1}(q, X_q)\,|\psi\rangle_P - \mathtt{W}_n(q, X_q)\,|\psi\rangle_P\| = 2\beta^2$. Since for infinite vectors the distance between the element of $\{\mathtt{W}_n(q, X_q)\,|\psi\rangle_P\}_n$ is constant, the sequence $\{\mathtt{W}_n(q, X_q)\,|\psi\rangle_P\}_n$ is not Cauchy, and for Theorem 7.1 the sequence $\{\mathtt{W}_n(q, X_q)\}_n$ is not strongly convergent. Thus, the unitary operator semantics is only able to capture all the finite approximations of infinite results, i.e., for all $n \in \mathbb{N}$, $\mathtt{W}_n(q, [\,s\,])$ gives an *observable* state from which a classical output can be extracted via measurement. The circuit representations of a general unbounded and a k-bounded quantum loop are depicted in Figure 7.3b. Finally,

the unitary semantics of the quantum while statement can be defined by:

$$[\![\,\texttt{qwhile}\ q\,\texttt{do}\ \{s\}\,]\!]_n = \mathtt{W}_n(q, [\![\,s\,]\!])$$

for any $n < \infty$, to model only quantum loops limited to a certain finite number of iterations n.

To give a mathematical meaning to recursively defined programs, we need to construct a more abstract domain that contains a mathematically well-defined object capable of representing infinite computational sequences. We will define such a domain and a denotational semantics for SL in Section 7.3.

7.2.2 Examples

We show how the operator $\mathtt{W}_n$ works by means of some examples.

First we consider $\texttt{qwhile}^n\ q\,\texttt{do}\ \{X(q)\}$ with the state $|\psi\rangle = 1/\sqrt{2}\,|0\ldots\rangle_T\,(|0\rangle_q + |1\rangle_q)$ as input:

$$\begin{aligned}
\mathtt{W}_0(q, X_q)\,|\psi\rangle &= 1/\sqrt{2}\,|0\ldots\rangle_T\,(|0\rangle_q + |1\rangle_q)\\
\mathtt{W}_1(q, X_q)\,|\psi\rangle &= 1/\sqrt{2}(|0\ldots\rangle_T\,|0\rangle_q + |10\ldots\rangle_T\,|0\rangle_q)\\
\mathtt{W}_2(q, X_q)\,|\psi\rangle &= 1/\sqrt{2}(|0\ldots\rangle_T\,|0\rangle_q + |10\ldots\rangle_T\,|0\rangle_q)\\
&\ldots
\end{aligned} \tag{7.6}$$

Note that with this input, the while loop fully terminates, and in fact, $\mathtt{W}_n(q, X_q)$ reaches a fixpoint.

Things are different when non-termination is involved, as in the loop $\texttt{qwhile}\ q\,\texttt{do}\ \{H(q)\}$ with the same input $|\psi\rangle$:

$$\begin{aligned}
\mathtt{W}_0(q, H_q)\,|\psi\rangle &= 1/\sqrt{2}\,|0\ldots\rangle_T\,(|0\rangle_q + |1\rangle_q)\\
\mathtt{W}_1(q, H_q)\,|\psi\rangle &= 1/\sqrt{2}\,|0\ldots\rangle_T\,|0\rangle_q + 1/2(|10\ldots\rangle_T\,|0\rangle_q - |10\ldots\rangle_T\,|1\rangle_q)\\
\mathtt{W}_2(q, H_q)\,|\psi\rangle &= 1/\sqrt{2}\,|0\ldots\rangle_T\,|0\rangle_q + 1/2\,|10\ldots\rangle_T\,|0\rangle_q\\
&\quad + 1/\sqrt{8}(|110\ldots\rangle_T\,|0\rangle_q - |110\ldots\rangle_T\,|1\rangle_q)\\
\mathtt{W}_3(q, H_q)\,|\psi\rangle &= 1/\sqrt{2}\,|0\ldots\rangle_T\,|0\rangle_q + 1/2\,|10\ldots\rangle_T\,|0\rangle_q\\
&\quad + 1/\sqrt{8}\,|110\ldots\rangle_T\,|0\rangle_q + 1/4(|1110\ldots\rangle_T\,|0\rangle_q - |1110\ldots\rangle_T\,|1\rangle_q)\\
\mathtt{W}_n(q, H_q)\,|\psi\rangle &= \Sigma_{i=0}^{n} 1/\sqrt{2^{i+1}}\,\left|1^{\otimes i}0\ldots\right\rangle_T|0\rangle_q - 1/\sqrt{2^{n+1}}\left|1^{\otimes n}0\ldots\right\rangle_T|1\rangle_q\,.
\end{aligned} \tag{7.7}$$

Since each t_j controls the execution of the j-th iteration, $t_j = 1$ indicates that the j-th iteration has been executed (see Figure 7.3b and Figure 7.2b). During each iteration of $\mathtt{W}_n$, the terminating part of the state—where q equals $|0\rangle$—is gradually increased. In contrast,

the portion corresponding to non-termination, where q equals $|1\rangle$, decreases but never fully reaches zero.

Finally, consider again the while loop $\texttt{qwhile}\ q\ \texttt{do}\ \{\texttt{skip}\}$, evaluated on $|0\ldots\rangle_T |1\rangle_q$ in Equation 7.5. Here for a divergent loop, each $\texttt{W}_n$ differs from the previous ones, reflecting that the unitary semantics corresponds to the partial results of the divergent loop computation.

7.3 Linear Semantics

To model infinite computations in quantum computing, we need to enlarge the domain of denotations so as to include an appropriate limit. To this purpose, we observe that a unitary operator is also bounded and, therefore, can be seen as an element of the Banach Space of bounded linear operators on $\mathscr{H}_P$. We will use this domain to define a denotational semantics for SL, which we call *linear semantics* to highlight the fact that it is more general (or abstract) than the unitary semantics introduced in Section 7.2.

We start by defining the linear semantics for each statement in SL, except for the **while** statement.

Definition 7.3. *Let $\mathscr{B}(\mathscr{H}_P)$ be the space of linear bounded operators from $\mathscr{H}_P$ to $\mathscr{H}_P$ equipped with the operator norm $\|A\| = \sup_{\||\psi\rangle\|=1} \|A|\psi\rangle\|$. The linear semantics is a function $[\![\cdot]\!] : s \to \mathscr{B}(\mathscr{H}_P)$, defined by:*

(1) $[\![\,\texttt{skip}\,]\!] = \mathbf{I}_{\mathscr{H}_P}$;
(2) $[\![\,U(\mathbf{q})\,]\!] = \mathbf{U}$, *where* $\mathbf{U} = \mathbf{I} \otimes U \otimes \mathbf{I}$, *i.e. the extension of* U *on* $\mathscr{H}_P$;
(3) $[\![\,s_1;\, s_2\,]\!] = [\![\,s_2\,]\!] \cdot [\![\,s_1\,]\!]$;

Note that the linear semantics of these statements is exactly the same as the unitary semantics defined in Definition 7.1. For the **while** statement, we need instead to introduce a new bounded linear operator, which will allow us to give a meaning also to infinite computations.

Let S be a bounded linear operator and $q \in Q_s$. We define the operator $\texttt{l}_n(q, S)$ by:

$$\begin{aligned} \texttt{l}_0(q, S) &= 0 \\ \texttt{l}_n(q, S) &= (\texttt{P}_{0_{t_1}} + \texttt{P}_{1_{t_1}} \cdot \mathcal{S}(\texttt{l}_{n-1}(q, S)) \cdot S)\mathbf{G}(q, 1), \end{aligned} \tag{7.8}$$

where 0 is the zero operator on $\mathscr{H}_P$, $\mathcal{S}$ produces the shift $t_n \to t_{n+1}$, and the controlled operation is represented by the operator defined in Proposition 7.1.

It is easy to see that for $n > 0$, the operator $\texttt{l}_n(q, S)$ is equivalent to the operator $\texttt{w}_n(q, S)$

as defined in Equation 7.2; in fact, $\mathtt{l}_n(q,S)$ is defined as the composition of the control-not gate, which evaluates the guard, and the operator corresponding to the composition of the semantics of the loop body S and $\mathtt{l}_{n-1}(q,S)$ shifted by $\mathcal{S}$.

Proposition 7.3. *For all $n \in \mathbb{N}$, if S is bounded then $\mathtt{l}_n(q,S)$ is bounded.*

Proof. Let's consider $\mathtt{l}_n(q,S)$ and a vector $|\psi\rangle$ such that $\||\psi\rangle\| = 1$:

$$\begin{aligned}
&\|(\mathtt{P}_{0_{t_1}} + \mathtt{P}_{1_{t_1}} \cdot \mathcal{S}(\mathtt{l}_{i-1}(q,S)) \cdot S)\mathbf{G}(q,1)\,|\psi\rangle\,\|^2 = \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{(since } \mathbf{G}(q,1) \text{ is unitary)} \\
&= \|(\mathtt{P}_{0_{t_1}} + \mathtt{P}_{1_{t_1}} \cdot \mathcal{S}(\mathtt{l}_{i-1}(q,S)) \cdot S)\,|\psi\rangle\,\|^2 = \\
&= \|\mathtt{P}_{0_{t_1}}\,|\psi\rangle + \mathtt{P}_{1_{t_1}} \cdot \mathcal{S}(\mathtt{l}_{i-1}(q,S)) \cdot S\,|\psi\rangle\,\|^2 = \\
&\qquad\qquad\qquad\qquad\qquad\qquad (\mathtt{P}_{0_{t_1}} \text{ and } \mathtt{P}_{1_{t_1}} \text{ are orthogonal}) \\
&= \|\mathtt{P}_{0_{t_1}}\,|\psi\rangle\,\|^2 + \|\mathtt{P}_{1_{t_1}}\mathcal{S}(\mathtt{l}_{i-1}(q,S))S)\,|\psi\rangle\,\|^2 \leq \\
&\qquad\qquad\qquad\qquad\qquad\qquad (\mathcal{S}(\mathtt{l}_{i-1}(q,S))S) \text{ is bounded)} \\
&\qquad\qquad \leq \|\mathtt{P}_{0_{t_1}}\,|\psi\rangle\,\|^2 + \|\mathtt{P}_{1_{t_1}}\,|\psi\rangle\,\|^2 = \\
&\qquad = \|(\mathtt{P}_{0_{t_1}} + \mathtt{P}_{1_{t_1}})\,|\psi\rangle\,\|^2 = \|\,|\psi\rangle\,\|^2 = 1.
\end{aligned}$$

Since $\|\mathtt{l}_n(q,S)\,|\psi\rangle\,\|^2 \leq 1$ also $\|\mathtt{l}_n(q,S)\,|\psi\rangle\,\| \leq 1$. □

Proposition 7.4. *The closed formula of Equation 7.8 is:*

$$\mathtt{L}_n(q,S) = \sum_{k=1}^{n}\left(\prod_{i=1}^{k-1}(\mathtt{P}_{1_{t_i}})\cdot\mathtt{P}_{0_{t_k}} \cdot \prod_{i=n-k}^{n-2}\big(\boldsymbol{G}(q,n-i)\cdot S\big)\cdot \boldsymbol{G}(q,1)\right). \tag{7.9}$$

Proof. If $n = 0$, $\mathtt{l}_0 = 0$ and $\mathtt{L} = \sum_{k=1}^{0}(\dots) = 0$. By inductive hypothesis,

$$\mathtt{l}_n = \sum_{k=1}^{n}(\prod_{i=1}^{k-1}(\mathtt{P}_{1_{t_i}}) \cdot \mathtt{P}_{0_{t_k}} \cdot \prod_{i=n-k}^{n-2}(\mathbf{G}(q,n-i)[\![\, s \,]\!]) \cdot \mathbf{G}(q,1)).$$

Let's consider $\mathtt{l}_{n+1} = (\mathtt{P}_{0_{t_1}} + \mathtt{P}_{1_{t_1}} \cdot \mathcal{S}(\mathtt{l}_n) \cdot [\![\, s \,]\!])\mathbf{G}(q,1)$. By inductive hypothesis $\mathtt{l}_n = \mathtt{L}_n$. First we compute $\mathcal{S}(\mathtt{L}_n)$, in particular,

$$\mathcal{S}(\mathtt{L}_n) = \sum_{k=1}^{n}(\prod_{i=1}^{k-1}(\mathtt{P}_{1_{t_{i+1}}}) \cdot \mathtt{P}_{0_{t_{k+1}}} \cdot \prod_{i=n-k}^{n-2}(\mathbf{G}(q,n-i+1)[\![\, s \,]\!]) \cdot \mathbf{G}(q,2)).$$

The initial and final values of the first product can be updated to remove the $+1$ from the

indices, resulting in:

$$\mathcal{S}(\mathtt{L}_n) = \sum_{k=1}^{n}(\prod_{i=2}^{(k+1)-1} (\mathtt{P}_{1_{t_i}}) \cdot \mathtt{P}_{0_{t_{k+1}}} \cdot \prod_{i=n-k}^{n-2} (\mathbf{G}(q,(n+1)-i)[\![\, s \,]\!]) \cdot \mathbf{G}(q,2)).$$

Then, the $k+1$ can be collected, and the summation indices can be updated, resulting in the following:

$$\mathcal{S}(\mathtt{L}_n) = \sum_{k=2}^{n+1}(\prod_{i=2}^{k-1}(\mathtt{P}_{1_{t_i}}) \cdot \mathtt{P}_{0_{t_k}} \cdot \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q,(n+1)-i)[\![\, s \,]\!]) \cdot \mathbf{G}(q,2)).$$

Now we sobstitude this equation in $\mathtt{l}_{n+1} = \mathtt{P}_{0_{t_1}} \cdot \mathbf{G}(q,1) + \mathtt{P}_{1_{t_1}} \cdot \mathcal{S}(\mathtt{L}_n) \cdot [\![\, s \,]\!] \cdot \mathbf{G}(q,1)$. Initially, we compute $\mathtt{l}' = \mathtt{P}_{1_{t_1}} \cdot \mathcal{S}(\mathtt{L}_n)$, in particular:

$$\begin{aligned}
\mathtt{l}' =& \mathtt{P}_{1_{t_1}} \big(\sum_{k=2}^{n+1} \prod_{i=2}^{k-1} (\mathtt{P}_{1_{t_i}}) \mathtt{P}_{0_{t_k}} \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q,(n+1)-i)[\![\, s \,]\!]) \mathbf{G}(q,2) \big) \\
=& \sum_{k=2}^{n+1} (\mathtt{P}_{1_{t_1}} \prod_{i=2}^{k-1} (\mathtt{P}_{1_{t_i}}) \mathtt{P}_{0_{t_k}} \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q,(n+1)-i)[\![\, s \,]\!]) \mathbf{G}(q,2)) \\
=& \sum_{k=2}^{n+1} (\prod_{i=1}^{k-1} (\mathtt{P}_{1_{t_i}}) \mathtt{P}_{0_{t_k}} \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q,(n+1)-i)[\![\, s \,]\!]) \mathbf{G}(q,2)).
\end{aligned}$$

Secondly, let's consider $\mathtt{l}'' = \mathtt{l}' \cdot [\![\, s \,]\!] \cdot \mathbf{G}(q,1)$, it result in:

$$\begin{aligned}
\mathtt{l}'' =& \Big(\sum_{k=2}^{n+1} \prod_{i=1}^{k-1} (\mathtt{P}_{1_{t_i}}) \mathtt{P}_{0_{t_k}} \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q,(n+1)-i)[\![\, s \,]\!]) \mathbf{G}(q,2) \Big) [\![\, s \,]\!] \mathbf{G}(q,1) = \\
=& \sum_{k=2}^{n+1} \prod_{i=1}^{k-1} (\mathtt{P}_{1_{t_i}}) \mathtt{P}_{0_{t_k}} \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q,(n+1)-i)[\![\, s \,]\!]) \mathbf{G}(q,2) [\![\, s \,]\!] \mathbf{G}(q,1).
\end{aligned}$$

If we consider the second product, we can include $\mathbf{G}(q,2)[\![\, s \,]\!]$ by updating the indices. Specifically, when $i = n-2$, $n+1-n+2 = 3$, therefore, by setting $n-1$ as the upper limit of the product, we can include $\mathbf{G}(q,2)[\![\, s \,]\!]$ in the product. So finally, we can write $\mathtt{l}''$ as:

$$\mathtt{l}'' = \sum_{k=2}^{n+1} \prod_{i=1}^{k-1} (\mathtt{P}_{1_{t_i}}) \mathtt{P}_{0_{t_k}} \prod_{i=n-(k-1)}^{n-1} (\mathbf{G}(q,(n+1)-i)[\![\, s \,]\!]) \mathbf{G}(q,1).$$

Finally, since:

$$\mathtt{l}_{n+1} = \mathtt{P}_{0_{t_1}}\mathbf{G}(q,1) + \mathtt{l}'' \text{ and}$$
$$(\prod_{i=1}^{k-1}(\mathtt{P}_{1_{t_i}})\mathtt{P}_{0_{t_k}} \prod_{i=n-(k-1)}^{n-1}(\mathbf{G}(q,(n+1)-i)[\![\, s \,]\!])\mathbf{G}(q,1))_{k=1} = \mathtt{P}_{0_{t_1}} \cdot \mathbf{G}(q,1),$$

we can write:

$$\mathtt{l}_{n+1} = \sum_{k=1}^{n+1}\prod_{i=1}^{k-1}(\mathtt{P}_{1_{t_i}})\mathtt{P}_{0_{t_k}} \prod_{i=(n+1)-k}^{n-1}(\mathbf{G}(q,(n+1)-i)[\![\, s \,]\!])\mathbf{G}(q,1),$$

i.e., we have writen $\mathtt{l}_{n+1}$ in the form of Equation 7.9. □

We can now define a chain of linear operators $\{\mathtt{L}_n(q,[\![\, s \,]\!])\}_n$ representing the linear semantics of all finite executions of the while loop and, therefore, all finite approximations of the semantics of the $\mathtt{qwhile}\ q\ \mathtt{do}\ \{s\}$ statement. The following proposition guarantees that this semantics is well defined.

Proposition 7.5. *For all $n \in \mathbb{N}$, $\mathtt{L}_n(q,[\![\, s \,]\!])$ is bounded.*

Proof. The proof follows by means of a structural induction on s and Proposition 7.3 □

We now show that the approximation chain $\{\mathtt{L}_n(q,[\![\, s \,]\!])\}_n$ has a limit in $\mathscr{B}(\mathscr{H}_P)$, and we will use it to define the semantics of $\mathtt{qwhile}\ q\ \mathtt{do}\ \{s\}$.

From Theorem 7.1, we know that if we prove that the sequence $\{\|\mathtt{L}_n(q,[\![\, s \,]\!])\|\}$ is bounded and that, for all $|\psi\rangle \in \mathscr{H}_P$, the sequence $\{\mathtt{L}_n(q,[\![\, s \,]\!])\,|\psi\rangle\}$ is a Cauchy sequence in $\mathscr{H}_P$, then we can conclude that the sequence $\{\mathtt{L}_n(q,[\![\, s \,]\!])\}_n$ has a limit. So, let's first prove that $\{\|\mathtt{L}_n(q,[\![\, s \,]\!])\|\}_n$ is bounded.

Proposition 7.6. *The sequence $\{\|\mathtt{L}_n(q,[\![\, s \,]\!])\|\}$ is bounded.*

Proof. From Proposition 7.5, we know that for all $n \in \mathbb{N}$, $\|\mathtt{L}_n(q,[\![\, s \,]\!])\| \leq 1$, thus the sequence $\{\|\mathtt{L}_n(q,[\![\, s \,]\!])\|\}$ is bounded above [Mat13]. □

We will now show that, for all $|\psi\rangle \in \mathscr{H}_P$, the sequence $\{\mathtt{L}_n(q,[\![\, s \,]\!])\,|\psi\rangle\}$ is Cauchy. To this purpose, we will provide some supporting lemmas.

Lemma 7.1. *For all $n > 0$, $\mathtt{L}_n(q,[\![\, s \,]\!]) = \sum_{i=1}^{n}(\mathtt{L}_i(q,[\![\, s \,]\!]) - \mathtt{L}_{i-1}(q,[\![\, s \,]\!]))$.*

Proof. In general, given a sequence A_n, we can prove that $A_n = \sum_{i=1}^{n}(A_i - A_{i-1}) + A_0$ by induction on n and a straightforward arithmetic simplifications. In our case, $\mathtt{L}_0(q, [\![s]\!]) = 0$, thus $\mathtt{L}_n(q, [\![s]\!]) = \sum_{i=1}^{n} (\mathtt{L}_i(q, [\![s]\!]) - \mathtt{L}_{i-1}(q, [\![s]\!]))$. □

Lemma 7.2. *For every $n \neq m$, $\mathtt{L}_n(q, [\![s]\!]) - \mathtt{L}_{n-1}(q, [\![s]\!])$ and $\mathtt{L}_m(q, [\![s]\!]) - \mathtt{L}_{m-1}(q, [\![s]\!])$ are orthogonal.*

Proof. From Equation 7.9, we can compute:

$$\mathtt{L}_n(q, [\![s]\!]) - \mathtt{L}_{n-1}(q, [\![s]\!]) = \prod_{i=1}^{n-1} \left(\mathtt{P}_{1_{t_i}}\right) \cdot \mathtt{P}_{0_{t_n}} \cdot \prod_{i=0}^{n-2} (\mathbf{G}(q, n-i) \cdot S) \cdot \mathbf{G}(q, 1),$$

$$\mathtt{L}_m(q, [\![s]\!]) - \mathtt{L}_{m-1}(q, [\![s]\!]) = \prod_{i=1}^{m-1} \left(\mathtt{P}_{1_{t_i}}\right) \cdot \mathtt{P}_{0_{t_m}} \cdot \prod_{i=0}^{m-2} (\mathbf{G}(q, n-i) \cdot S) \cdot \mathbf{G}(q, 1).$$

Clearly, if $n \neq m$,

$$\prod_{i=1}^{n-1} \left(\mathtt{P}_{1_{t_i}}\right) \mathtt{P}_{0_{t_n}} \text{ and } \prod_{i=1}^{m-1} \left(\mathtt{P}_{1_{t_i}}\right) \mathtt{P}_{0_{t_m}}$$

are orthogonal, and thus

$$\mathtt{L}_n(q, [\![s]\!]) - \mathtt{L}_{n-1}(q, [\![s]\!]) \text{ and } \mathtt{L}_m(q, [\![s]\!]) - \mathtt{L}_{m-1}(q, [\![s]\!])$$

are orthogonal. □

Lemma 7.3. *For every n, m, if $n \leq m$, then $\|\mathtt{L}_n(q, [\![s]\!])\,|\psi\rangle\| \leq \|\mathtt{L}_m(q, [\![s]\!])\,|\psi\rangle\|$.*

Proof. From Equation 7.9, we observe that if $n \leq m$, the summation corresponding to $\mathtt{L}_m(q, [\![s]\!])$ contains all the terms of $\mathtt{L}_n(q, [\![s]\!])$ plus additional ones. Thus, if $\mathtt{L}_n(q, [\![s]\!])\,|\psi\rangle = \sum \alpha_i\,|e_i\rangle$ and $\mathtt{L}_m(q, [\![s]\!])\,|\psi\rangle = \sum \beta_i\,|e_i\rangle$, where $|e_i\rangle$ is a vector in the standard basis of $\mathscr{H}_P$, then $\forall i,\ \alpha_i \neq 0 \Rightarrow \beta_i = \alpha_i$. In other words, $\mathtt{L}_n(q, [\![s]\!])\,|\psi\rangle$ is a sub-state of $\mathtt{L}_m(q, [\![s]\!])\,|\psi\rangle$ since $\mathtt{L}_m(q, [\![s]\!])$ projects the state on 'more basis vectors' than $\mathtt{L}_n(q, [\![s]\!])$. □

Theorem 7.2. *For all vectors $|\psi\rangle \in \mathscr{H}_P$, the sequence $\{\mathtt{L}_n(q, [\![s]\!])\,|\psi\rangle\}_n$ is Cauchy.*

Proof. By Lemma 7.1, $\mathtt{L}_n(q, [\![s]\!])\,|\psi\rangle = \Sigma_{i=1}^{n}(\mathtt{L}_i(q, [\![s]\!]) - \mathtt{L}_{i-1}(q, [\![s]\!]))\,|\psi\rangle$. By Lemma 7.2 and [HZ08, Exercise 2] we have:

$$\|\mathtt{L}_n(q, [\![s]\!])\,|\psi\rangle\|^2 = \Sigma_{i=1}^{n}\|(\mathtt{L}_i(q, [\![s]\!]) - \mathtt{L}_{i-1}(q, [\![s]\!]))\,|\psi\rangle\|^2.$$

Let us consider the sequence of partial sums $\{\Sigma_{j=1}^{n}\|(\mathtt{L}_i(q, [\![s]\!]) - \mathtt{L}_{i-1}(q, [\![s]\!]))\,|\psi\rangle\|^2\}_n$. By Proposition 7.5 and Lemma 7.3, we know that all partial sums are bounded and

that the sequence of partial sums is increasing. Since the sequence of partial sums is both increasing and bounded above, it must converge [Mat13]. This implies that the infinite sum $\Sigma_{j=1}^{\infty}\|(\mathtt{L}_i(q,[\![\, s \,]\!]) - \mathtt{L}_{i-1}(q,[\![\, s \,]\!]))\,|\psi\rangle\,\|^2$ also converges [Spi08]. We have that $\Sigma_{i=1}^{\infty}\|(\mathtt{L}_i(q,[\![\, s \,]\!]) - \mathtt{L}_{i-1}(q,[\![\, s \,]\!]))\,|\psi\rangle\,\|^2$ converges, and for all i:

$$\|(\mathtt{L}_i(q,[\![\, s \,]\!]) - \mathtt{L}_{i-1}(q,[\![\, s \,]\!]))\,|\psi\rangle\,\|^2 > 0.$$

Thus, $\lim_{i\to\infty}\left(\|(\mathtt{L}_i(q,[\![\, s \,]\!]) - \mathtt{L}_{i-1}(q,[\![\, s \,]\!]))\,|\psi\rangle\,\|^2\right) = 0$, which implies that

$$\lim_{i\to\infty}\left(\|\mathtt{L}_i(q,[\![\, s \,]\!])\,|\psi\rangle - \mathtt{L}_{i-1}(q,[\![\, s \,]\!])\,|\psi\rangle\,\|\right) = 0.$$

This ensures that the sequence $\{\mathtt{L}_i(q,[\![\, s \,]\!])\,|\psi\rangle\}_i$ is a Cauchy sequence in $\mathscr{H}_P$. □

Having shown that, for all $|\psi\rangle \in \mathscr{H}_P$, $\{\mathtt{L}_n(q,[\![\, s \,]\!]\,|\psi\rangle\}_n$ is a Cauchy sequence, by Theorem 7.1 we can conclude that the sequence $\{\mathtt{L}_n(q,[\![\, s \,]\!])\}$ has a limit. Thus, we use that limit to define the semantics of the quantum loop:

$$[\![\, \mathtt{qwhile}\ q\ \mathtt{do}\ \{s\} \,]\!] = \lim_{n\to\infty} \mathtt{L}_n(q,[\![\, s \,]\!]).$$

Moreover, the following proposition shows that this limit corresponds to a well-defined linear operator on $\mathscr{H}_P$.

Proposition 7.7. *Let $\{|e_i\rangle\}_i$ be the set of standard basis of $\mathscr{H}_P$, and let $\mathscr{L}(q,[\![\, s \,]\!])$ be the linear operator defined as $\mathscr{L}(q,[\![\, s \,]\!])\,|e_i\rangle = \lim_{n\to\infty}(\mathtt{L}_n(q,[\![\, s \,]\!])\,|e_i\rangle)$, $\forall\,|e_i\rangle$. Then*

$$\mathscr{L}(q,[\![\, s \,]\!]) = \lim_{n\to\infty}\{\mathtt{L}_n(q,[\![\, s \,]\!])\}_n$$

.

Proof. By definition, for all basis $|e_i\rangle$ of $\mathscr{H}_P$, $\mathscr{L}(q,[\![\, s \,]\!])\,|e_i\rangle = \lim_{n\to\infty}\mathtt{L}_n(q,[\![\, s \,]\!])\,|e_i\rangle$, thus $\|\mathtt{L}_n(q,[\![\, s \,]\!])\,|e_i\rangle - \mathscr{L}(q,[\![\, s \,]\!])\,|e_i\rangle\,\| \to 0$ as $n \to \infty$. Therefore, for all $|\psi\rangle \in \mathscr{H}_P$, $\|\mathtt{L}_n(q,[\![\, s \,]\!])\,|\psi\rangle - \mathscr{L}(q,[\![\, s \,]\!])\,|\psi\rangle\,\| \to 0$ as $n \to \infty$. □

7.4 Relation between Unitary and Linear Semantics

So far, we have introduced two semantics:

- The unitary semantics models exactly the behavior of quantum computations.
- The linear semantics allows us to define a fixpoint and therefore the semantics of infinite loops.

In this section, we investigate the relationship between these two semantics.

Let us consider the examples introduced in subsection 7.2.2. We can construct the semantics of the program `qwhile` q `do` $\{X(q)\}$, starting from the state $|\psi\rangle = {}^1\!/\!\sqrt{2}\,|0\ldots\rangle_T\,(|0\rangle_q + |1\rangle_q)$, as follows:

$$\begin{aligned}
&\mathtt{L}_0(q, X_q)\,|\psi\rangle = 0\\
&\mathtt{L}_1(q, X_q)\,|\psi\rangle = {}^1\!/\!\sqrt{2}\,|0\ldots\rangle_T\,|0\rangle_q\\
&\mathtt{L}_2(q, X_q)\,|\psi\rangle = {}^1\!/\!\sqrt{2}(|0\ldots\rangle_T\,|0\rangle_q + |10\ldots\rangle_T\,|0\rangle_q)\\
&\mathtt{L}_3(q, X_q)\,|\psi\rangle = {}^1\!/\!\sqrt{2}(|0\ldots\rangle_T\,|0\rangle_q + |10\ldots\rangle_T\,|0\rangle_q)
\end{aligned} \tag{7.10}$$

We see that we have 'collected' the sub-state that corresponds to the terminating execution, and since the program is fully terminating, the fixpoint is a state with a norm equal to 1. Comparing the examples in the previous sections (Equation 7.6), it is easy to see that when the loop terminates, the two semantics coincide, specifically $\mathtt{L}_{n+1} = \mathtt{W}_n$. Here, the linear semantics is 'behind' the unitary one because the latter includes the component that corresponds to executions that 'keep going.'

Now consider the program `qwhile` q `do` $\{H(q)\}$ and the construction of its semantics:

$$\begin{aligned}
&\mathtt{L}_0(q, H_q)\,|\psi\rangle = 0\\
&\mathtt{L}_1(q, H_q)\,|\psi\rangle = {}^1\!/\!\sqrt{2}\,|0\ldots\rangle_T\,|0\rangle_q\\
&\mathtt{L}_2(q, H_q)\,|\psi\rangle = {}^1\!/\!\sqrt{2}\,|0\ldots\rangle_T\,|0\rangle_q + {}^1\!/\!2\,|10\ldots\rangle_T\,|0\rangle_q\\
&\mathtt{L}_3(q, H_q)\,|\psi\rangle = {}^1\!/\!\sqrt{2}\,|0\ldots\rangle_T\,|0\rangle_q + {}^1\!/\!2\,|10\ldots\rangle_T\,|0\rangle_q + {}^1\!/\!\sqrt{8}\,|110\ldots\rangle_T\,|0\rangle_q)\\
&\mathtt{L}_n(q, H_q)\,|\psi\rangle = \sum_{i=1}^{n} {}^1\!/\!\sqrt{2^n}\,\left|1^{\otimes i}0\ldots\right\rangle_T\,|0\rangle_q\,.
\end{aligned} \tag{7.11}$$

Here, we see that in each approximation $\mathtt{L}_n$, we obtain a state with a norm less than 1, which corresponds to the part of the execution that terminates in at most $n-1$ iterations. If we compare the linear semantics in Equation 7.11 with the corresponding unitary semantics in Equation 7.7, we observe that the linear semantics has a 'missing' part in the output states—specifically, the portion of the state representing the execution that 'keeps going'.

Finally, consider the while loop `qwhile` q `do` $\{skip\}$, evaluated on $|0\ldots\rangle_T\,|1\rangle_q$:

$$\begin{aligned}
&\mathtt{L}_0(q, \mathbf{I})\,|0\ldots\rangle_T\,|1\rangle_q = 0\\
&\mathtt{L}_1(q, \mathbf{I})\,|0\ldots\rangle_T\,|1\rangle_q = \mathtt{P}_{0_{t_1}}\,|10\ldots\rangle_T\,|1\rangle_q = 0\\
&\mathtt{L}_2(q, \mathbf{I})\,|0\ldots\rangle_T\,|1\rangle_q = \mathtt{P}_{0_{t_1}}\,|10\ldots\rangle_T\,|1\rangle_q + \mathtt{P}_{1_{t_1}}\mathtt{P}_{0_{t_2}}\,|110\ldots\rangle_T\,|1\rangle_q = 0.
\end{aligned} \tag{7.12}$$

When dealing with a loop that diverges for the whole quantum state, the linear semantics

in Equation 7.12 evaluates to 0. On the other hand, the unitary semantics (Equation 7.5) computes all partial executions of the divergent loop.

More generally, in all examples, the unitary operator $\mathtt{W}_n$ returns the portion of the state corresponding to executions that terminate after less than n iterations, along with the partial results of computations that are still ongoing. In contrast, the linear semantics can separate the terminating portion of the computation. In fact, by examining the definition of $\mathtt{W}_n$ (Equation 7.3), we can identify two key components:

$$\sum_{k=1}^{n}\left(\prod_{i=1}^{k-1}(\mathtt{P}_{1_{t_i}})\cdot\mathtt{P}_{0_{t_k}}\cdot\prod_{i=n-k}^{n-2}\left(\mathbf{G}(q,n-i)\cdot S\right)\cdot\mathbf{G}(q,1)\right),$$

that correspond exactly to linear semantics equation (Equation 7.9), and the final term is given by:

$$\prod_{i=1}^{n}\mathtt{P}_{1_{t_i}}\cdot\prod_{i=0}^{n-1}(S\cdot\mathbf{G}(q,n-i)).$$

The first part corresponds to computations where the guard becomes 0 after the $(n-1)$-th execution, and corresponds to the executions that terminate after at most $n-1$ iterations. The second part represents the non-terminating portion, where the guard remains true, meaning that the loop is still in progress. Thus, we can formalize the relation between the two semantics as follows.

Theorem 7.3. *Given a qubit $q \in Q_s$, and a linear operator S on $\mathscr{H}_T \otimes \mathscr{H}_{Q_s}$ that does not act on $\mathscr{H}_T$, for all $n \in \mathbb{N}$,*

$$\mathtt{L}_n(q,S) = \mathtt{P}_{0_{t_n}}\mathtt{W}_n(q,S).$$

Proof. The proof follows directly from Equation 7.3 and Equation 7.9. □

Corollary 7.1. *For all $|\psi\rangle \in \mathscr{H}_T \otimes \mathscr{H}_{Q_s}$, let $|\phi\rangle = \mathtt{W}_n(q,S)\,|\psi\rangle$. If $t_n = |0\rangle$ in $|\phi\rangle$, then $\mathtt{L}_n(q,S) = \mathtt{W}_n(q,S)$.*

Corollary 7.2. *For all $|\psi\rangle \in \mathscr{H}_T \otimes \mathscr{H}_{Q_s}$, for all $n \in \mathbb{N}$, $\|\mathtt{L}_n(q,S)\,|\psi\rangle\| \leq \|\mathtt{W}_n(q,S)\,|\psi\rangle\|$, i.e. the norm of the result of linear semantics is less or equal than the norm of the result of the unitary one.*

In other words, the norm $\|[\![\,\mathtt{qwhile}\ q\ \mathtt{do}\ \{s\}\,]\!]\,|\psi\rangle\|$ gives a measure of the portion of the state that diverges. Specifically, if the loop diverges, the semantics will be equal to the zero operator (as in Equation 7.12). If both terminating and non-terminating executions are in a superposition, the linear semantics will return a state with a norm less than one, and the missing norm value provides a measure of non-termination. For instance, in Equation 7.11

we see that for all $n \in \mathbb{N}$, $\|\mathtt{L}(q, H_q)\,|\psi\rangle\,\| < 1$, and also the norm of the limit, applied to $|\psi\rangle$ will be less than one. On the other hand, if the program terminates on every branch, the linear semantics correspond to the unitary one, and the norm $\|[\![\,\mathtt{qwhile}\ q\ \mathtt{do}\ \{s\}\,]\!]\,|\psi\rangle\,\|$ will be equal to 1 (as we see in Equation 7.10).

Corollary 7.3. *For all $|\psi\rangle_{Q_s} \in \mathscr{H}_{Q_s}$ and for all $n \in \mathbb{N}$, let*

$$|\phi_n\rangle = [\,\mathtt{qwhile}\ q\ \mathtt{do}\ \{s\}\,]_n(|0\rangle_T\,|\psi\rangle_{Q_s}).$$

If there exists $N \in \mathbb{N}$ such that, for all $n > N$, the qubit q in $|\phi_n\rangle$ is always in state $|0\rangle$, then

$$[\![\,\mathtt{qwhile}\ q\ \mathtt{do}\ \{s\}\,]\!](|0\rangle_T\,|\psi\rangle_{Q_s}) = [\,\mathtt{qwhile}\ q\ \mathtt{do}\ \{s\}\,]_n(|0\rangle_T\,|\psi\rangle_{Q_s}) \quad \textit{for all } n > N.$$

The linear semantics, therefore, returns sub-states of the results of the unitary semantics, i.e., possibly non-normalized states obtained by applying a projector to the outcome of the unitary semantics. By discarding part of the result, the linear semantics makes it possible to define a limit and, consequently, to assign meaning to infinite behaviors. This gives us the intuition that the linear semantics is actually *projecting away* something from the unitary semantics. The following theorem formalizes this intuition about the relation between the two semantics.

Theorem 7.4 (Adequacy)**.** *Let S be a SL program, and let $q \in \mathscr{H}_{Q_S}$. Then, for all $n \in \mathbb{N}$,*

$$\mathtt{L}_n(q, S) = \mathbb{P}(\mathtt{W}_n(q, S)),$$

where $\mathbb{P}$ is the orthogonal projection of $\mathscr{H}_S$ onto $\mathscr{H}_{Q_S}$ defined by $|0\rangle\langle 0| \otimes \mathbb{I}_{Q_S \setminus q} \otimes \mathbb{I}_T$.

The proof is a direct application of a stronger result in functional analysis, which we state below (see [Kud22] for a complete treatment).

Theorem 7.5. *Let H be a Hilbert space, and let T be a contraction (i.e. a bounded linear operator with norm ≤ 1) on H. There is a Hilbert space Y containing H as a subspace and a unitary transformation U on Y such that for all $n \in \mathbb{N}$*

$$T^n = PU^n \text{ and } (T^*)^n = PU^{-n},$$

where P is the orthogonal projection of Y onto H.

Proof of Theorem 7.4. From Theorem 7.5 with $H = \mathscr{H}_{Q_S}$, $Y = \mathscr{H}_S$, T^n as the linear semantics $\mathtt{L}_n(q, S)$, and U as the unitary semantics $\mathtt{W}_n(q, S)$, by noting that $\mathscr{H}_{Q_S}$ is a subspace of $\mathscr{H}_S = \mathscr{H}_T \otimes \mathscr{H}_{Q_s}$, and that $\mathbb{P}$ is the orthogonal projection of $\mathscr{H}_S$ onto $\mathscr{H}_{Q_S}$. □

Corollary 7.4. *The linear semantics capture the I/O behaviour of the unitary semantics.*

7.5 A Complete Example: Quantum Gambler Walk

A quantum walk [Aha+01; Amb+01] is the quantum-mechanical counterpart of a classical random walk [Pea05]. In a classical random walk, a particle occupies definite states and moves randomly between them according to stochastic rules. In contrast, a quantum walker evolves deterministically: its position is described by a superposition of states, its unitary evolution is governed by quantum interference, and randomness appears only at measurement. Unlike classical random walks, quantum walks do not converge to limiting distributions; interference makes them spread faster or slower, a property that underlies their usefulness in algorithm design. In this section, we show how quantum walks can be modelled using our semantics framework.

Let us consider the simple quantum walk presented in [Amb+01], called the Hadamard walk. The Hilbert space of the Hadamard walk is $\mathscr{H}_W = \mathscr{H}_d \otimes \mathscr{H}_p$, where $\mathscr{H}_d$ is a two-dimensional Hilbert space representing the direction ($|0\rangle$ is left while $|1\rangle$ is right) and $\mathscr{H}_p$ is an infinite-dimensional space representing the position of the walker. In particular, we have the state $|n\rangle \in \mathscr{H}_p . n \in \mathbb{Z}\}$ as the position holder. One step of the Hadamard walk is described by the following program:

$$H(d);\ S(d,p);$$

where $S(d,p) = |0\rangle\langle 0|_d \otimes L_p + |1\rangle\langle 1|_d \otimes R_p$, R is the unitary operator on $\mathscr{H}_p$, such that $R\,|n\rangle = |n+1\rangle$ and corresponds to the unitary that performs one step to the right, while $L\,|n\rangle = |n-1\rangle$ and corresponds to the unitary that performs one step to the left. For instance, if we start from the initial state $|0\rangle_d\,|0\rangle_p$, the first steps are:

$$\begin{aligned}
&(1)\ |0\rangle_d\,|0\rangle_p \xrightarrow{H(d)} {}^1\!/\!_{\sqrt{2}}\,(|0\rangle_d + |1\rangle_d)\,|0\rangle_p \xrightarrow{S(d,p)} {}^1\!/\!_{\sqrt{2}}\left(|0,-1\rangle_{d,p} + |1,1\rangle_{d,p}\right)\\
&(2)\ \xrightarrow{H(d)} \tfrac{1}{2}\left((|0\rangle_d + |1\rangle_d)\,|-1\rangle_p + (|0\rangle_d - |1\rangle_d)\,|1\rangle_p\right) \to\\
&\quad\xrightarrow{S(d,p)} \tfrac{1}{2}\left(|0,-2\rangle_{d,p} + |1,0\rangle_{d,p} + |0,0\rangle_{d,p} - |1,2\rangle_{d,p}\right)\\
&(3)\ \xrightarrow{H(d)} \tfrac{1}{2\sqrt{2}}\left(|0,-2\rangle + |1,-2\rangle + |0,0\rangle - |1,0\rangle + |0,0\rangle + |1,0\rangle - |0,2\rangle - |1,2\rangle\right)_{d,p} \to\\
&\quad\xrightarrow{S(d,p)} \tfrac{1}{2\sqrt{2}}\left(|0,-3\rangle_{d,p} + |1,-1\rangle_{d,p} + 2\,|0,-1\rangle_{d,p} - |0,1\rangle_{d,p} + |1,3\rangle_{d,p}\right).
\end{aligned}$$

In the third step, we see how the interference works in quantum walks, cancelling some walks while joining others. In fact, the Hadamard walk exhibits a position distribution that is nearly uniform over the range $[-t/\sqrt{2},\ t/\sqrt{2}]$ after t steps, in contrast to the classical random walk, whose position typically lies within a distance $O(\sqrt{t})$ from the origin with high probability [Amb+01].

Example: Quantum Gambler Walk

Let us consider a variation of the previous walk. We consider a variation of the Hadamard walk, where we have one absorbing boundary: in this walk, we restrict the walker's position to remain non-negative. This walk can be seen as a quantum analogue of the classical random walk called *gambler ruin* [Coo09]. In fact, in the gambler's ruin, the position can be interpreted as the amount of money the walker has. When it reaches 0, the walker can no longer gamble and therefore remains still.

In [Amb+01], the authors present a variation of the one-dimensional Hadamard walk with a left absorbing boundary, called the *semi-infinite quantum walk*. This walk is defined on the half-line with an absorbing boundary at position $p = 0$. After each Hadamard step, the system is measured to check whether it has reached the boundary: if the measurement outcome is $p = 0$, the walker is absorbed and the process terminates; otherwise, the walk continues. The process can be described by the following small program:

$$\begin{array}{l}\texttt{while}\ \mathit{measure}(p) \neq 0\ \texttt{do}\ \{\\ \quad H(d); S(d,p);\\ \}\end{array}$$

This construction models a quantum random walk with absorption, where the boundary check is explicitly performed through measurement. In this version, the termination condition is observed via measurement at each step of the walk; thus, the computation does not proceed in full coherence, and the result is a probability distribution rather than a superposition of executions.

If we wish to introduce a semi-infinite quantum walk whose evolution remains fully coherent, we need a condition depending on the position of the walker; therefore, a quantum `while`-loop is required. To distinguish this coherent version from the previous one, we call it the *Quantum Gambler Walk*. It can be described by the following small program:

$$\begin{array}{l}\texttt{qwhile}\ p \neq |0\rangle\ \texttt{do}\ \{\\ \quad H(d); S(d,p);\\ \}\end{array}$$

Let us define again $\mathscr{H}_W = \mathscr{H}_d \otimes \mathscr{H}_p$, where $\mathscr{H}_d$ is a two-dimensional Hilbert space representing the direction ($|0\rangle$ stands for "left" while $|1\rangle$ stands for "right"), and $\mathscr{H}_p$ is an infinite-dimensional Hilbert space representing the position of the walker. In addition, we need the space of temporary qubits to represent the quantum loop; hence, the entire program space

is given by $\mathscr{H}_T \otimes \mathscr{H}_W$. Let $W = [\, H(d); \texttt{qif}\, d\, \texttt{do}\, \{R(p)\}\, \texttt{else}\, \{L(p)\}\,]$ be the semantics of the loop body. The unitary semantics of this program can be expressed by the sequence of operators $\texttt{W}_i(p, W)$. Consider the first few steps of the walk, starting from the initial state $|\psi_0\rangle = |00\ldots\rangle_T\, |0\rangle_d\, |2\rangle_p$:

$$
\begin{aligned}
\texttt{W}_0(p,W)\,|\psi_0\rangle &= |\psi_0\rangle \\
\texttt{W}_1(p,W)\,|\psi_0\rangle &= \tfrac{1}{\sqrt{2}}\,|10\ldots\rangle_T \left(\,|0,1\rangle_{p,d} + |1,3\rangle_{p,d}\,\right) \\
\texttt{W}_2(p,W)\,|\psi_0\rangle &= \tfrac{1}{2}\,|110\ldots\rangle_T \left(\,|0,0\rangle_{p,d} + |1,2\rangle_{p,d} + |0,2\rangle_{p,d} - |1,4\rangle_{p,d}\,\right) \\
\texttt{W}_3(p,W)\,|\psi_0\rangle &= \tfrac{1}{2}\,|110\ldots\rangle_T\, |0,0\rangle_{p,d} + \tfrac{1}{2\sqrt{2}}\,|1110\ldots\rangle_T \left(|0,1\rangle_{p,d} - |1,3\rangle_{p,d} + |0,1\rangle_{p,d} + \right. \\
&\qquad \left. + |1,3\rangle_{p,d} \;- |0,3\rangle_{p,d} \;+ |1,5\rangle_{p,d}\,\right)
\end{aligned}
$$

In the last step, we can observe the effect of quantum interference in the walk.

We can also analyse how the linear semantics evolves and how it captures the portion of the walk that halts, i.e., when the walker reaches position 0. Formally, the linear semantics is defined by the sequence $\texttt{L}_i(p, W)$, and in particular:

$$
\begin{aligned}
\texttt{L}_0(p,W)\,|\psi_0\rangle &= 0 \\
\texttt{L}_1(p,W)\,|\psi_0\rangle &= 0 \\
\texttt{L}_2(p,W)\,|\psi_0\rangle &= \tfrac{1}{2}\;|110\ldots\rangle_T\, |0,0\rangle_{p,d} \\
\texttt{L}_3(p,W)\,|\psi_0\rangle &= \tfrac{1}{2}\;|110\ldots\rangle_T\, |0,0\rangle_{p,d} \\
\texttt{L}_4(p,W)\,|\psi_0\rangle &= \tfrac{1}{2}\;|110\ldots\rangle_T\, |0,0\rangle_{p,d} + \tfrac{1}{2}\;|11110\ldots\rangle_T\, |0,0\rangle_{p,d} \\
\texttt{L}_5(p,W)\,|\psi_0\rangle &= \tfrac{1}{2}\;|110\ldots\rangle_T\, |0,0\rangle_{p,d} + \tfrac{1}{2}\;|11110\ldots\rangle_T\, |0,0\rangle_{p,d} \\
\texttt{L}_6(p,W)\,|\psi_0\rangle &= \tfrac{1}{2}\;|110\ldots\rangle_T\, |0,0\rangle_{p,d} + \tfrac{1}{2}\;|11110\ldots\rangle_T\, |0,0\rangle_{p,d} + \tfrac{1}{8}\;|1111110\ldots\rangle_T\, |0,0\rangle_{p,d}
\end{aligned}
$$

In the case of the semi-infinite quantum walk, when the walker starts from position $p = 1$, the probability that it exits to the left is $2/\pi$, which is significantly lower than in the classical case, where it tends to 1 [Amb+01]. This reduction arises from quantum interference, which persists in the semi-infinite quantum walk even though measurements are performed, as the states with $p = 0$ and $p \neq 0$ remain orthogonal. Similarly, in our case, interference also occurs. Empirically, we observe that the probability of being in a state with $p = 0$, starting from $p = 1$, tends to $2/\pi$ while starting from $p = 2$, the probability of measuring $p = 0$ after n iterations (as $n \to \infty$) approaches 0.55.

7.6 Halting in Quantum Loops

The issue of quantum-controlled iterations has long been investigated in connection with the quantum Turing machine (QTM) and the QRAM model. Several works [LP98; MO05; Mye97; Shi02; Son08] highlighted the fact that a halting qubit cannot be consistently defined within a QTM. Myers [Mye97] was probably the first one to point out a flaw in Deutsch's original definition of a universal QTM [DP85]: when the input is in superposition, the different computation branches may terminate asynchronously, with some of them possibly never halting. As a result, one cannot check termination by simply measuring a flag qubit, since the measurement would collapse the superposition of branches and hence alter the computation itself. Bernstein and Vazirani addressed this problem in [BV97] by changing the definition of QTM, enforcing an explicit halting scheme: all branches must be synchronized, so that if one branch halts, every other branch halts at the same time. This insight was the first to clearly reveal that when a quantum computation evolves coherently into multiple branches, they must either all terminate simultaneously or diverge together. Such an observation has later been consolidated by further studies [LP98; MO05; Mye97; Shi02; Son08], showing that this limitation is not a peculiarity of the QTM formalism, but rather an inherent feature of quantum computation itself, directly connected to the unitarity of quantum evolution. Moreover, the issue of having execution branches of different lengths arises not only in the context of infinite computations but also when analyzing finite computations, as noted in recent works on quantum register machines and control-based models [ZY25; YVC24].

Our framework captures this intrinsic constraint of quantum computation from an algebraic perspective. The unitary semantics exactly represents the execution of a quantum program as the evolution of the corresponding quantum system. By definition, all branches have the same length: at each execution step, i.e., each step of the circuit implementing the program, all branches in superposition evolve together. When a branch has already terminated, its subsequent operations are simply replaced by identities through controlled operators. This becomes evident when considering the chain of unitary operators generated by the unfolding of a quantum loop. The halting problem is then reflected in the fact that this unitary chain has no limit. The existence of a limit of the linear semantics is made possible by giving up unitarity, specifically by removing the component of the semantics that enforces synchronization of partial executions. Indeed, the linear operators forming the chain of finite approximations are only able to compute the sub-executions that have already terminated. In a sense, the key step that enables us to define a limit operator for infinite computations is precisely that linear operators allow different branches to terminate at different times, while the non-terminating ones are discarded at each stage. Thus, in order to obtain a well-defined

limit operator, we relaxed the constraint first identified by Myers [Mye97] in the original definition of quantum Turing machines. Of course, this relaxation comes at a cost: we lose unitarity. The resulting limit operator no longer describes the exact quantum evolution of the system but only part of what would arise by letting a quantum computation evolve indefinitely.

7.7 Discussion and Related Work

We have introduced a denotational semantics for quantum programs, which approximates the unitary behaviour of the programs by means of linear operators acting on possibly non-normalized states corresponding to the programs' finite results.

Various approaches to the problem of modelling the control flow in a quantum program have been introduced in the literature on the design and implementation of quantum programming languages. They can be grouped as follows.

Probabilistic control flow The initial works on quantum program semantics focused on languages with quantum data and classical probabilistic control, primarily based on measurement operators. In [Sel04], Selinger introduced the basic notations, theories, and conventions for a quantum programming language with measurement-based probabilistic control flow, called QPL. He provided a denotational semantics for QPL by associating each program with a superoperator (a completely positive map that is not necessarily trace-preserving) in a finite-dimensional Hilbert space, represented as a morphism in a CPO-enriched traced monoidal category.

In [Per08a], Perdrix extended this work by introducing a complete partial order (CPO) over admissible transformations, i.e., multisets of linear operators, and using it to define a denotational semantics for a simple quantum imperative language similar to QPL. He demonstrated that this semantics is an exact abstraction of Selinger's semantics.

In a series of works [Fen+07; YF10; YYF12; Yin12], Ying et al. explored a quantum while language with measurement-based probabilistic control flow. They defined a denotational semantics in terms of maps between density operators, generalizing Selinger's results to infinite-dimensional Hilbert spaces. In these works, the denotational semantics is constructed using the CPO of superoperators acting on partial density operators. Finally, in [Yin16, Chapter 3], Ying further developed this domain to define the semantics of a quantum language with recursion and measurement-based probabilistic control flow, providing a more comprehensive framework for reasoning about quantum programs.

Quantum Control All the previous works focused on language with probabilistic control flow, thus their semantics is probabilistic and no superposition is introduced between possible executions of the programs.

A first form of quantum control was introduced in Altenkirch and Grattage's functional language QML [AG05], a first-order functional language on finite types equipped with a categorical semantics capturing only finite quantum computations. In [Lam+08], Lampis et al. introduced nQML, a simplified version of QML with simpler control constructs and a denotational semantics based on density matrices and unitary transformations, still capturing only finite computations.

The issue of modelling quantum control was formally addressed by Badescu and Panangaden in [BP15]. They extended the QPL programming language [Sel04] by introducing a quantum if-statement and provided a denotational semantics for this extension based on Kraus decomposition. However, they observed that the semantics of quantum case statements is not monotone with respect to Selinger's order, concluding that the Selinger framework is inadequate for modelling the semantics of a quantum language containing both quantum control and probabilistic control flow.

A formal definition of quantum imperative language with quantum control flow was introduced by Ying et al. in [YYF12; YYF14], where they define the QuGCL language, i.e., a language with both quantum control flow and measurement-based control flow but no recursion. A semantics for this language is given in terms of a new mathematical tool called the guarded composition of operator-valued functions, where operator-valued functions are defined using the Kraus operator-sum representation [NC10, Chapter 8]. However, this semantics is not compositional.

More recently, in [Val22], Valiron has reviewed the use of quantum control in the λ-calculus, explaining the difference between superposition of terms and superposition of data in the various formulations.

With respect to these previous works, our work focuses on a language whose control flow is purely quantum. By removing measurements, we can define the semantics directly in terms of unitary operators, thus providing a more suitable framework for modelling quantum control without the additional complications of combining quantum control with probabilistic control flow. We show that even in a measurement-free setting, it is impossible to define a limit for the sequence of unitary operators. This highlights the inherent challenges in modelling quantum recursion within a purely unitary framework. Moreover, the absence of measurements allows us to better characterize the difficulties of handling quantum recursion and to introduce a linear semantics that addresses these issues by incorporating suitable

approximations.

The quantum control had also been considered from a more practical point of view. In [YVC24], Yuan et al. studied the problem of automatically compiling self-controlled quantum operations, minimizing the use of extra temporary variables and avoiding that these extra temporary variables are entangled with the rest of the variables. In particular, they formalize the conditions under which this compilation is possible. Their compilation technique can be seen as a particular case of our unitary semantics. Inspired by Yuan et al., Zhang and Ying [ZY25] propose a quantum architecture that supports quantum control flow and finite quantum recursion.

Finally, Barsse et al. [BPP26] introduce a recursive quantum programming language that incorporates both measurement and coherent quantum control. In their framework, coherent quantum control enables arbitrary operations and is equipped with both operational and denotational semantics based on default evolution and vacuum extensions [CK19; Kri+20]. However, recursion is restricted to probabilistic control flow, and the language does not explicitly account for loops or recursion driven by coherent quantum control.

Quantum Recursion Introducing a construct for quantum control flow in a programming language naturally leads to the concepts of quantum recursion or quantum loops. An initial idea of quantum recursion, based on using an infinite set of external coins, was informally discussed in [YYF12].

In [SVV18], Sabry et al. extend a classical, typed, reversible language that includes lists and fixpoints to a quantum setting. The resulting quantum language is equipped with an operational semantics based on the principles of algebraic λ-calculi. This work proves that it is possible to represent a quantum program with recursion with a unitary operator, only if we are able to construct the fixpoint of the recursive call by means of a finite unfolding of the recursive calls. As we do not impose any restriction on the quantum loop, our semantics is more general.

In his PhD thesis [And22], Andrés-Martínez introduces a quantum while language similar to ours but equipped with a categorical semantics. The thesis extends Haghverdi's unique decomposition categories—originally introduced to model iteration in classical computation—by addressing their incompatibility with the quantum settings. This generalisation establishes connections to topological groups and leads to a hierarchy of categories enriched with infinitary addition and convergence criteria. Building on this foundation, the execution formula is shown to define a valid categorical trace even over categories of quantum processes on infinite-dimensional Hilbert spaces. This approach, however, relies on a computational model that is not immediately referable to quantum circuits. In defining our semantics,

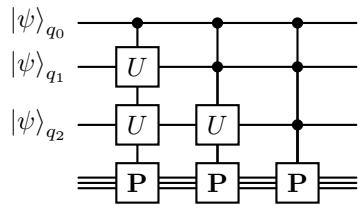


Figure 7.4: $\mathbf{P} := \texttt{qif}\ q\ \texttt{do}\ \{U(q); \mathbf{P}\}$ circuit

we, instead, refer to the standard computation model of quantum computing; in fact, our unitary semantics is directly implementable on a quantum computer.

In [Yin16; Yin14], Ying explores a problem similar to ours, considering a recursive quantum language. This work has a stronger similarity with our approach than the other works mentioned above, although the implementation of the self-controlled operation is, in a sense, 'dual' with respect to our model. In fact, instead of making a quantum copy of the guard using a `CX` gate, Ying proposes to prepare and carry along the program an infinite number of copies of identical qubits to represent the guard. However, since we are trying to represent in a unitary way an operation that inherently cannot be unitary, this difference is merely a design choice to address the unitarity constraint.

In this setting, recursion is achieved with programs of the form $\mathbf{P} := \texttt{qif}\ q\ \texttt{do}\ \{U(q); \mathbf{P}\}$, which can be visually represented in Figure 7.4. It can be seen that each recursive call consumes a copy of the guard variable, and to achieve infinite recursion, we need an infinite number of copies of the same guard variable. For defining their semantics, Ying et al. employ a formalism from quantum physics related to multi-particle systems, specifically Fock spaces [Shc13] and second quantization [Bar03]. In particular, they consider free Fock spaces, i.e., Hilbert spaces that describe quantum states with a variable number of indistinguishable particles, constructed as $\oplus_{n=1}^{\infty} \mathscr{H}^{\otimes n}$, i.e., the direct sum of tensor powers of the single-particle Hilbert space $\mathscr{H}$. Using this formulation, Ying defines a Complete Partial Order (CPO), which orders the operators within the Fock space based on the number of guard copies that these operators act on. As a result, the semantics of a recursive program demonstrates a monotonically continuous order, which, in turn, allows for the existence of a fixed point.

Both ours and Ying's semantics share the key feature of utilising an infinite number of qubits to perform while loops and defining recursive unitary operators, as illustrated in Figure 7.2b and Figure 7.4. In fact, we can choose to formulate our approach within Ying's Fock space semantics or, conversely, describe Ying's semantics using our Unitary/Linear framework. In the latter case, we can specifically adopt the formulation introduced by

Ying in [Yin14, Section 6.4], mapping the Fock space operator to a unitary operator on the program variables space as in Figure 7.4, and subsequently to a linear operator, in the same way as in Section 7.3.

Chapter 8

Conclusion

The results presented in this thesis contribute to laying the theoretical and methodological foundations for a principled study of quantum program analysis. By extending core concepts of abstract interpretation and denotational semantics to the quantum realm, this thesis demonstrates how classical reasoning techniques can be systematically adapted to accommodate the non-classical nature of quantum computation. The resulting frameworks provide a unified view of analysis and verification for quantum programs, encompassing both operational concerns—such as variable lifetime and uncomputation—and high-level semantic aspects, such as entanglement, robustness, and quantum control.

Beyond their individual technical results, the proposed approaches collectively point toward a broader methodological shift: rather than treating quantum programming as a mere extension of classical paradigms, they advocate for dedicated abstractions and semantics that reflect the physical and mathematical specificities of quantum information. In this sense, the thesis not only advances the state of the art in static analysis and formal verification for quantum programs, but also contributes to the long-term goal of establishing a rigorous, compositional theory of quantum software.

Ultimately, this work aims to bridge the gap between quantum theory, language design, and program verification. As the field transitions from experimental prototypes to scalable, programmable devices, the need for sound and automated reasoning tools will become increasingly pressing. The ideas developed here provide a foundation for such tools, supporting the development of quantum software that is not only expressive and efficient but also provably correct and robust to the uncertainties inherent in quantum computation.

Bibliography

[Abd+25] Parosh Aziz Abdulla et al. 'Verifying Quantum Circuits with Level-Synchronized Tree Automata'. In: *Proc. ACM Program. Lang.* 9.POPL (Jan. 2025). DOI: 10.1145/3704868. URL: https://doi.org/10.1145/3704868.

[ADM24a] Nicola Assolini, Alessandra Di Pierro and Isabella Mastroeni. 'Abstracting Entanglement'. In: *Proceedings of the 10th ACM SIGPLAN International Workshop on Numerical and Symbolic Abstract Domains, NSAD 2024, Pasadena, CA, USA, 22 October 2024*. Ed. by Vincenzo Arceri and Michele Pasqua. ACM, 2024, pp. 34–41. DOI: 10.1145/3689609.3689998. URL: https://doi.org/10.1145/3689609.3689998.

[ADM24b] Nicola Assolini, Alessandra Di Pierro and Isabella Mastroeni. 'Static Analysis of Quantum Programs'. In: *Static Analysis - 31st International Symposium, SAS 2024, Pasadena, CA, USA, October 20-22, 2024, Proceedings*. Ed. by Roberto Giacobazzi and Alessandra Gorla. Vol. 14995. Lecture Notes in Computer Science. Springer, 2024, pp. 1–25. DOI: 10.1007/978-3-031-74776-2"˙1. URL: https://doi.org/10.1007/978-3-031-74776-2%5C˙1.

[ADM25] Nicola Assolini, Alessandra Di Pierro and Isabella Mastroeni. 'A Static Analysis of Entanglement'. In: *Verification, Model Checking, and Abstract Interpretation - 26th International Conference, VMCAI 2025, Denver, CO, USA, January 20-21, 2025, Proceedings, Part II*. Ed. by Shankaranarayanan Krishna, Sriram Sankaranarayanan and Ashutosh Trivedi. Vol. 15530. Lecture Notes in Computer Science. Springer, 2025, pp. 50–71. DOI: 10.1007/978-3-031-82703-7"˙3. URL: https://doi.org/10.1007/978-3-031-82703-7%5C˙3.

[AG04] Scott Aaronson and Daniel Gottesman. 'Improved Simulation of Stabilizer Circuits'. In: *CoRR* quant-ph/0406196 (2004). URL: http://arxiv.org/abs/quant-ph/0406196.

[AG05] Thorsten Altenkirch and Jonathan Grattage. 'A Functional Quantum Programming Language'. In: *20th IEEE Symposium on Logic in Computer Science*

(LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings. IEEE Computer Society, 2005, pp. 249–258. DOI: 10.1109/LICS.2005.1. URL: https://doi.org/10.1109/LICS.2005.1.

[AG20] Matthew Amy and Vlad Gheorghiu. 'staq—A full-stack quantum processing toolkit'. In: *Quantum Science and Technology* 5.3 (2020), p. 034016.

[Aha+01] Dorit Aharonov et al. 'Quantum walks on graphs'. In: *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*. STOC '01. Hersonissos, Greece: Association for Computing Machinery, 2001, pp. 50–59. ISBN: 1581133499. DOI: 10.1145/380752.380758. URL: https://doi.org/10.1145/380752.380758.

[Aho+06] Alfred V. Aho et al. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[AL25] Matthew Amy and Joseph Lunderville. 'Linear and Non-linear Relational Analyses for Quantum Program Optimization'. In: *Proc. ACM Program. Lang.* 9.POPL (Jan. 2025). DOI: 10.1145/3704873. URL: https://doi.org/10.1145/3704873.

[Ale+19] Gadi Aleksandrowicz et al. *Qiskit: An Open-source Framework for Quantum Computing*. Version 0.7.2. Jan. 2019. DOI: 10.5281/zenodo.2562111. URL: https://doi.org/10.5281/zenodo.2562111.

[Amb+01] Andris Ambainis et al. 'One-dimensional quantum walks'. In: *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*. STOC '01. Hersonissos, Greece: Association for Computing Machinery, 2001, pp. 37–49. ISBN: 1581133499. DOI: 10.1145/380752.380757. URL: https://doi.org/10.1145/380752.380757.

[AMK15] Parent Alex, Roetteler Martin and Svore Krysta M. 'Reversible circuit compilation with space constraints'. In: (2015). arXiv: 1510.00377 [quant-ph].

[AMM14] Matthew Amy, Dmitri Maslov and Michele Mosca. 'Polynomial-time T-depth optimization of Clifford+ T circuits via matroid partitioning'. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.10 (2014), pp. 1476–1489.

[AMM25] Nicola Assolini, Alessandra Marzari Luca Di Pierro and Isabella Mastroeni. 'Formal Verification of Variational Quantum Circuits'. In: *CoRR* abs/2507.10635 (2025). DOI: 10.48550/ARXIV.2507.10635. arXiv: 2507.10635. URL: https://doi.org/10.48550/arXiv.2507.10635.

[Amy18] Matthew Amy. 'Towards Large-Scale Functional Verification of Universal Quantum Circuits'. In: *Proceedings of the 15th International Conference on Quantum Physics and Logic (QPL 2018)*. Vol. 287. EPTCS. 2018. DOI: 10.4204/EPTCS.287.1. URL: https://doi.org/10.4204/EPTCS.287.1.

[Amy19] Matthew Amy. 'Formal methods in quantum circuit design'. PhD thesis. University of Waterloo, 2019.

[And22] Pablo Andrés-Martínez. 'Unbounded loops in quantum programs: categories and weak while loops'. In: *arXiv preprint arXiv:2212.05371* (2022).

[AP25] Nicola Assolini and Alessandra Di Pierro. 'A Denotational Semantics for Quantum Loops'. In: *CoRR* abs/2506.23320 (2025). DOI: 10.48550/ARXIV.2506.23320. arXiv: 2506.23320. URL: https://doi.org/10.48550/arXiv.2506.23320.

[APM26] Nicola Assolini, Alessandra Di Pierro and Isabella Mastroeni. 'Challenges in Quantum Programs Analysis'. In: *International Journal on Software Tools for Technology Transfer* (2026). ISSN: 1433-2787. DOI: 10.1007/s10009-026-00845-1. URL: https://doi.org/10.1007/s10009-026-00845-1.

[ARS17] Matthew Amy, Martin Roetteler and Krysta M. Svore. 'Verified Compilation of Space-Efficient Reversible Circuits'. In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*. Ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10427. Lecture Notes in Computer Science. Springer, 2017, pp. 3–21. DOI: 10.1007/978-3-319-63390-9"˙1. URL: https://doi.org/10.1007/978-3-319-63390-9%5C˙1.

[Bar+95] Adriano Barenco et al. 'Elementary gates for quantum computation'. In: *Phys. Rev. A* 52 (5 Nov. 1995), pp. 3457–3467. DOI: 10.1103/PhysRevA.52.3457. URL: https://link.aps.org/doi/10.1103/PhysRevA.52.3457.

[Bar03] Michel Baranger. 'Many-Body Problems and Quantum Field Theory: An Introduction'. In: *Physics Today* 56.4 (Apr. 2003), pp. 69–70. ISSN: 0031-9228. DOI: 10.1063/1.1580057. eprint: https://pubs.aip.org/physicstoday/article-pdf/56/4/69/11139124/69"˙1"˙online.pdf. URL: https://doi.org/10.1063/1.1580057.

[Beh+25] Joshua A. C. Behler et al. 'Static Analysis and Transformation for Quantum Programming Languages'. In: *IEEE Software* 42.5 (2025), pp. 58–64. DOI: 10.1109/MS.2025.3566634.

[Ben73] C. H. Bennett. 'Logical Reversibility of Computation'. In: *IBM Journal of Research and Development* 17.6 (1973), pp. 525–532. DOI: 10.1147/rd.176.0525.

[Ber+18] Ville Bergholm et al. 'PennyLane: Automatic differentiation of hybrid quantum-classical computations'. In: *CoRR* abs/1811.04968 (2018). arXiv: 1811.04968. URL: http://arxiv.org/abs/1811.04968.

[Bia+17] Jacob Biamonte et al. 'Quantum machine learning'. In: *Nature* 549.7671 (2017), pp. 195–202.

[Bic+20] Benjamin Bichsel et al. 'Silq: a high-level quantum language with safe uncomputation and intuitive semantics'. In: *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2020. London, UK: Association for Computing Machinery, 2020, pp. 286–300. ISBN: 9781450376136. DOI: 10.1145/3385412.3386007. URL: https://doi.org/10.1145/3385412.3386007.

[Bic+23] Benjamin Bichsel et al. 'Abstraqt: Analysis of Quantum Circuits via Abstract Stabilizer Simulation'. In: *Quantum* 7 (2023), p. 1185. DOI: 10.22331/Q-2023-11-20-1185. URL: https://doi.org/10.22331/q-2023-11-20-1185.

[Boy+00] P.Oscar Boykin et al. 'A new universal and fault-tolerant quantum basis'. In: *Information Processing Letters* 75.3 (2000), pp. 101–107. ISSN: 0020-0190. DOI: https://doi.org/10.1016/S0020-0190(00)00084-3. URL: https://www.sciencedirect.com/science/article/pii/S0020019000000843.

[BP15] Costin Badescu and Prakash Panangaden. 'Quantum Alternation: Prospects and Problems'. In: *Proceedings 12th International Workshop on Quantum Physics and Logic, QPL 2015, Oxford, UK, July 15-17, 2015*. EPTCS 195 (2015). Ed. by Chris Heunen, Peter Selinger and Jamie Vicary, pp. 33–42. DOI: 10.4204/EPTCS.195.3. URL: https://doi.org/10.4204/EPTCS.195.3.

[BPP26] Kathleen Barsse, Romain Péchoux and Simon Perdrix. 'Quantum Control and General Recursion beyond the Unitary Case'. working paper or preprint. Mar. 2026. URL: https://hal.science/hal-05538043.

[Bru+23] Roberto Bruni et al. 'A Correctness and Incorrectness Program Logic'. In: *J. ACM* 70.2 (Mar. 2023). ISSN: 0004-5411. DOI: 10.1145/3582267. URL: https://doi.org/10.1145/3582267.

[Bun+18] Rudy R Bunel et al. 'A unified view of piecewise linear neural network verification'. In: *Advances in Neural Information Processing Systems* 31 (2018).

[Bun+20] Rudy Bunel et al. 'Branch and bound for piecewise linear neural network verification'. In: *Journal of Machine Learning Research* 21.42 (2020), pp. 1–39.

[BV97] Ethan Bernstein and Umesh Vazirani. 'Quantum Complexity Theory'. In: *SIAM Journal on Computing* 26.5 (1997), pp. 1411–1473. DOI: 10.1137/S0097539796300921. eprint: https://doi.org/10.1137/S0097539796300921. URL: https://doi.org/10.1137/S0097539796300921.

[BW92] Charles H. Bennett and Stephen J. Wiesner. 'Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states'. In: *Phys. Rev. Lett.*

69 (20 Nov. 1992), pp. 2881–2884. DOI: 10.1103/PhysRevLett.69.2881. URL: https://link.aps.org/doi/10.1103/PhysRevLett.69.2881.

[CC77] Patrick Cousot and Radhia Cousot. 'Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints'. In: *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL '77. Los Angeles, California: Association for Computing Machinery, 1977, pp. 238–252. ISBN: 9781450373500. DOI: 10.1145/512950.512973. URL: https://doi.org/10.1145/512950.512973.

[CC79] Patrick Cousot and Radhia Cousot. 'Systematic design of program analysis frameworks'. In: *Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL '79. San Antonio, Texas: Association for Computing Machinery, 1979, pp. 269–282. ISBN: 9781450373579. DOI: 10.1145/567752.567778. URL: https://doi.org/10.1145/567752.567778.

[CD09] Bob Coecke and Ross Duncan. 'Interacting Quantum Observables: Categorical Algebra and Diagrammatics'. In: *CoRR* abs/0906.4725 (2009). arXiv: 0906.4725. URL: http://arxiv.org/abs/0906.4725.

[CD25] Andrea Colledan and Ugo Dal Lago. 'Flexible Type-Based Resource Estimation in Quantum Circuit Description Languages'. In: *Proc. ACM Program. Lang.* 9.POPL (Jan. 2025). DOI: 10.1145/3704883. URL: https://doi.org/10.1145/3704883.

[CDM11] Michael L. Collard, Michael John Decker and Jonathan I. Maletic. 'Lightweight Transformation and Fact Extraction with the srcML Toolkit'. In: *11th IEEE Working Conference on Source Code Analysis and Manipulation, SCAM 2011, Williamsburg, VA, USA, September 25-26, 2011*. IEEE Computer Society, 2011, pp. 173–184. DOI: 10.1109/SCAM.2011.19. URL: https://doi.org/10.1109/SCAM.2011.19.

[CDM13] Michael L. Collard, Michael John Decker and Jonathan I. Maletic. 'srcML: An Infrastructure for the Exploration, Analysis, and Manipulation of Source Code: A Tool Demonstration'. In: *2013 IEEE International Conference on Software Maintenance, Eindhoven, The Netherlands, September 22-28, 2013*. IEEE Computer Society, 2013, pp. 516–519. DOI: 10.1109/ICSM.2013.85. URL: https://doi.org/10.1109/ICSM.2013.85.

[Cer+21] Marco Cerezo et al. 'Variational quantum algorithms'. In: *Nature Reviews Physics* 3.9 (2021), pp. 625–644.

[CFM24] Yanbin Chen, Innocenzo Fulginiti and Christian B. Mendl. 'Probabilistic Circuit Model'. In: *2024 IEEE International Conference on Quantum Computing and*

Engineering (QCE). Vol. 02. 2024, pp. 508–509. DOI: 10.1109/QCE60285.2024.10379.

[CFM25] Yanbin Chen, Innocenzo Fulginiti and Christian B. Mendl. 'Optimization Framework for Reducing Mid-circuit Measurements and Resets'. In: *Computational Science - ICCS 2025 Workshops - 25th International Conference, Singapore, Singapore, July 7-9, 2025, Proceedings, Part V*. Ed. by Maciej Paszynski, Amanda S. Barnard and Yongjie Jessica Zhang. Vol. 15911. Lecture Notes in Computer Science. Springer, 2025, pp. 150–164. DOI: 10.1007/978-3-031-97570-7"˙13. URL: https://doi.org/10.1007/978-3-031-97570-7%5C˙13.

[Cha+21] Clément Chareton et al. 'An Automated Deductive Verification Framework for Circuit-building Quantum Programs'. In: *Programming Languages and Systems. ESOP 2021*. Ed. by Nobuko Yoshida. Vol. 12648. Lecture Notes in Computer Science. Springer, Cham, 2021, pp. 156–182. DOI: 10.1007/978-3-030-72019-3˙6. URL: https://doi.org/10.1007/978-3-030-72019-3˙6.

[Che+23a] Yu-Fang Chen et al. 'An Automata-Based Framework for Verification and Bug Hunting in Quantum Circuits'. In: *Proc. ACM Program. Lang.* 7.PLDI (June 2023). DOI: 10.1145/3591270. URL: https://doi.org/10.1145/3591270.

[Che+23b] Yu-Fang Chen et al. 'AutoQ: An Automata-Based Quantum Circuit Verifier'. In: *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III*. Ed. by Constantin Enea and Akash Lal. Vol. 13966. Lecture Notes in Computer Science. Springer, 2023, pp. 139–153. DOI: 10.1007/978-3-031-37709-9"˙7. URL: https://doi.org/10.1007/978-3-031-37709-9%5C˙7.

[Che+25] Yu-Fang Chen et al. 'AutoQ 2.0: From Verification of Quantum Circuits to Verification of Quantum Programs'. In: *Tools and Algorithms for the Construction and Analysis of Systems - 31st International Conference, TACAS 2025, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, ON, Canada, May 3-8, 2025, Proceedings, Part III*. Ed. by Arie Gurfinkel and Marijn Heule. Vol. 15698. Lecture Notes in Computer Science. Springer, 2025, pp. 87–108. DOI: 10.1007/978-3-031-90660-2"˙5. URL: https://doi.org/10.1007/978-3-031-90660-2%5C˙5.

[Cir25] Cirq Developers. *Cirq*. Version 1.6.1. 13th Aug. 2025. DOI: 10.5281/zenodo.4062499. URL: https://quantumai.google/cirq.

[CK19] Giulio Chiribella and Hlér Kristjánsson. 'Quantum Shannon theory with superpositions of trajectories'. In: *Proceedings of the Royal Society A: Mathem-*

atical, Physical and Engineering Sciences 475.2225 (May 2019), p. 20180903. ISSN: 1364-5021. DOI: 10.1098/rspa.2018.0903. eprint: https://royalsocietypublishing.org/rspa/article-pdf/doi/10.1098/rspa.2018.0903/346401/rspa.2018.0903.pdf. URL: https://doi.org/10.1098/rspa.2018.0903.

[CMS25] Yanbin Chen, Christian B. Mendl and Helmut Seidl. 'Dead Gate Elimination'. In: *Computational Science - ICCS 2025 - 25th International Conference, Singapore, July 7-9, 2025, Proceedings, Part III*. Ed. by Michael H. Lees et al. Vol. 15905. Lecture Notes in Computer Science. Springer, 2025, pp. 135–150. DOI: 10.1007/978-3-031-97632-2"˙10. URL: https://doi.org/10.1007/978-3-031-97632-2%5C˙10.

[Coo09] J. L. Coolidge. 'The Gambler's Ruin'. In: *Annals of Mathematics* 10.4 (1909), pp. 181–192. ISSN: 0003486X, 19398980. URL: http://www.jstor.org/stable/1967408 (visited on 23/10/2025).

[Cou+06] Patrick Cousot et al. 'Combination of Abstractions in the ASTRÉE Static Analyzer'. In: *Advances in Computer Science - ASIAN 2006. Secure Software and Related Issues, 11th Asian Computing Science Conference, Tokyo, Japan, December 6-8, 2006, Revised Selected Papers*. Ed. by Mitsu Okada and Ichiro Satoh. Vol. 4435. Lecture Notes in Computer Science. Springer, 2006, pp. 272–300. DOI: 10.1007/978-3-540-77505-8"˙23. URL: https://doi.org/10.1007/978-3-540-77505-8%5C˙23.

[Cou21a] Patrick Cousot. 'Dynamic interval analysis by abstract interpretation'. en. In: *Formal Methods in Outer Space* 13065 (2021). Ed. by Ezio Bartocci, Yliès Falcone and Martin Leucker. Series Title: Lecture Notes in Computer Science, pp. 61–86. DOI: 10.1007/978-3-030-87348-6˙4. URL: https://link.springer.com/10.1007/978-3-030-87348-6˙4 (visited on 23/04/2025).

[Cou21b] Patrick Cousot. *Principles of Abstract Interpretation*. MIT Press, 2021, pp. 1–819. ISBN: 9780262044905.

[CS23] Yanbin Chen and Yannick Stade. 'Quantum Constant Propagation'. In: *Static Analysis - 30th International Symposium, SAS 2023, Cascais, Portugal, October 22-24, 2023, Proceedings*. Ed. by Manuel V. Hermenegildo and José F. Morales. Vol. 14284. Lecture Notes in Computer Science. Springer, 2023, pp. 164–189. DOI: 10.1007/978-3-031-44245-2"˙9. URL: https://doi.org/10.1007/978-3-031-44245-2%5C˙9.

[DD21] Evandro Chagas Ribeiro Da Rosa and Rafael De Santiago. 'Ket Quantum Programming'. In: *J. Emerg. Technol. Comput. Syst.* 18.1 (Oct. 2021). ISSN:

1550-4832. DOI: 10.1145/3474224. URL: https://doi.org/10.1145/3474224.

[DGM19] Sumanth Dathathri, Sicun Gao and Richard M Murray. 'Inverse abstraction of neural networks using symbolic interpolation'. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 3437–3444.

[Di +18] Alessandra Di Pierro et al. 'Homological analysis of multi-qubit entanglement'. In: *Europhysics Letters* 123.3 (Sept. 2018), p. 30006. DOI: 10.1209/0295-5075/123/30006. URL: https://dx.doi.org/10.1209/0295-5075/123/30006.

[Din+20] Yongshan Ding et al. 'SQUARE: Strategic Quantum Ancilla Reuse for Modular Quantum Programs via Cost-Effective Uncomputation'. In: *47th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2020, Virtual Event / Valencia, Spain, May 30 - June 3, 2020*. IEEE, 2020, pp. 570–583. DOI: 10.1109/ISCA45697.2020.00054. URL: https://doi.org/10.1109/ISCA45697.2020.00054.

[DK18] Pierre-Luc Dallaire-Demers and Nathan Killoran. 'Quantum generative adversarial networks'. In: *Phys. Rev. A* 98 (1 June 2018), p. 012324. DOI: 10.1103/PhysRevA.98.012324. URL: https://link.aps.org/doi/10.1103/PhysRevA.98.012324.

[DP85] David Deutsch and Roger Penrose. 'Quantum theory, the Church–Turing principle and the universal quantum computer'. In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.1818 (1985), pp. 97–117. DOI: 10.1098/rspa.1985.0070. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rspa.1985.0070. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1985.0070.

[End+21] Suguru Endo et al. 'Hybrid quantum-classical algorithms and quantum error mitigation'. In: *Journal of the Physical Society of Japan* 90.3 (2021), p. 032001.

[Fen+07] Yuan Feng et al. 'Proof rules for the correctness of quantum programs'. In: *Theoretical Computer Science* 386.1 (2007), pp. 151–166. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2007.06.011. URL: https://www.sciencedirect.com/science/article/pii/S0304397507004926.

[Fer+22] Claudio Ferrari et al. 'Complete verification via multi-neuron relaxation guided branch-and-bound'. In: *ICRL* (2022).

[Fer04] Jérôme Feret. 'Static Analysis of Digital Filters'. In: *Programming Languages and Systems, 13th European Symposium on Programming, ESOP 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software,*

ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings. Ed. by David A. Schmidt. Vol. 2986. Lecture Notes in Computer Science. Springer, 2004, pp. 33–48. DOI: 10.1007/978-3-540-24725-8“˙4. URL: https://doi.org/10.1007/978-3-540-24725-8%5C˙4.

[Geh+18] Timon Gehr et al. ‘AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation’. In: *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 2018, pp. 3–18. DOI: 10.1109/SP.2018.00058. URL: https://doi.org/10.1109/SP.2018.00058.

[GFY21] Ji Guan, Wang Fang and Mingsheng Ying. ‘Robustness Verification of Quantum Classifiers’. In: *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I*. Ed. by Alexandra Silva and K. Rustan M. Leino. Lecture Notes in Computer Science. Springer, 2021, pp. 151–174. DOI: 10.1007/978-3-030-81685-8“˙7. URL: https://doi.org/10.1007/978-3-030-81685-8%5C˙7.

[GH71] Irene Gargantini and Peter Henrici. ‘Circular arithmetic and the determination of polynomial zeros’. In: *Numerische Mathematik* 18 (1971), pp. 305–320.

[GMP24] Roberto Giacobazzi, Isabella Mastroeni and Elia Perantoni. ‘Adversities in Abstract Interpretation - Accommodating Robustness by Abstract Interpretation’. In: *ACM Trans. Program. Lang. Syst.* 46.2 (2024), p. 5. DOI: 10.1145/3649309.

[Got98] Daniel Gottesman. ‘The Heisenberg representation of quantum computers’. In: *arXiv preprint quant-ph/9807006* (1998).

[Gra+19] Edward Grant et al. ‘An initialization strategy for addressing barren plateaus in parametrized quantum circuits’. In: *Quantum* 3 (Dec. 2019), p. 214. ISSN: 2521-327X. DOI: 10.22331/q-2019-12-09-214. URL: https://doi.org/10.22331/q-2019-12-09-214.

[Gre+13a] Alexander S. Green et al. ‘An Introduction to Quantum Programming in Quipper’. In: *Reversible Computation - 5th International Conference, RC 2013, Victoria, BC, Canada, July 4-5, 2013. Proceedings*. Ed. by Gerhard W. Dueck and D. Michael Miller. Vol. 7948. Lecture Notes in Computer Science. Springer, 2013, pp. 110–124. DOI: 10.1007/978-3-642-38986-3“˙10. URL: https://doi.org/10.1007/978-3-642-38986-3%5C˙10.

[Gre+13b] Alexander S. Green et al. ‘Quipper: A Scalable Quantum Programming Language’. In: *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI ’13. Seattle, Washington, USA: Association for Computing Machinery, 2013, pp. 333–342. ISBN: 9781450320146.

DOI: 10.1145/2491956.2462177. URL: https://doi.org/10.1145/2491956.2462177.

[GRS00] Roberto Giacobazzi, Francesco Ranzato and Francesca Scozzari. 'Making abstract interpretations complete'. In: *J. ACM* 47.2 (2000), pp. 361–416. DOI: 10.1145/333979.333989. URL: https://doi.org/10.1145/333979.333989.

[Hav+19] Vojtech Havlicek et al. 'Supervised learning with quantum enhanced feature spaces'. en. In: *Nature* 567.7747 (Mar. 2019). arXiv:1804.11326 [quant-ph], pp. 209–212. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-019-0980-2. URL: http://arxiv.org/abs/1804.11326 (visited on 26/05/2025).

[HH25] Kengo Hirata and Chris Heunen. 'Qurts: Automatic Quantum Uncomputation by Affine Types with Lifetime'. In: *Proc. ACM Program. Lang.* 9.POPL (Jan. 2025). DOI: 10.1145/3704842. URL: https://doi.org/10.1145/3704842.

[Hie+21] Kesha Hietala et al. 'A verified optimizer for Quantum circuits'. In: *Proc. ACM Program. Lang.* 5.POPL (Jan. 2021). DOI: 10.1145/3434318. URL: https://doi.org/10.1145/3434318.

[Hon15] Kentaro Honda. 'Analysis of Quantum Entanglement in Quantum Programs using Stabilizer Formalism'. In: *Proceedings 12th International Workshop on Quantum Physics and Logic, QPL 2015, Oxford, UK, July 15-17, 2015*. Ed. by Chris Heunen, Peter Selinger and Jamie Vicary. Vol. 195. EPTCS. 2015, pp. 262–272. DOI: 10.4204/EPTCS.195.19. URL: https://doi.org/10.4204/EPTCS.195.19.

[HR24] Po-Wei Huang and Patrick Rebentrost. *Post-variational quantum neural networks*. 2024. arXiv: 2307.10560 [quant-ph]. URL: https://arxiv.org/abs/2307.10560.

[HZ08] Teiko Heinosaari and Mario Ziman. 'Guide to mathematical concepts of quantum theory'. In: *Acta Physica Slovaca* 58.4 (Aug. 2008), pp. 487–674. DOI: 10.2478/v10155-010-0091-y. arXiv: 0810.3536 [quant-ph].

[Itt+21] David Ittah et al. 'Enabling dataflow optimization for quantum programs'. In: *arXiv preprint arXiv:2101.11030* (2021).

[Jav+14] Ali JavadiAbhari et al. 'ScaffCC: a framework for compilation and analysis of quantum computing programs'. In: *Proceedings of the 11th ACM Conference on Computing Frontiers*. CF '14. Cagliari, Italy: Association for Computing Machinery, 2014. ISBN: 9781450328708. DOI: 10.1145/2597917.2597939. URL: https://doi.org/10.1145/2597917.2597939.

[JHG19] E.R. Johnston, N. Harrigan and M. Gimeno-Segovia. *Programming Quantum Computers: Essential Algorithms and Code Samples*. O'Reilly Media, Incorporated, 2019.

[JW96] W. Just and M. Weese. *Discovering Modern Set Theory. I: The Basics*. Discovering Modern Set Theory. American Mathematical Society, 1996. ISBN: 9780821802663.

[KKB23] Maximilian Kaul, Alexander Küchler and Christian Banse. 'A Uniform Representation of Classical and Quantum Source Code for Static Code Analysis'. In: *IEEE International Conference on Quantum Computing and Engineering, QCE 2023, Bellevue, WA, USA, September 17-22, 2023*. Ed. by Brian La Cour, Lia Yeh and Marek Osinski. IEEE, 2023, pp. 1013–1019. DOI: 10.1109/QCE57702.2023.00115. URL: https://doi.org/10.1109/QCE57702.2023.00115.

[KLM06] Phillip Kaye, Raymond Laflamme and Michele Mosca. *An introduction to quantum computing*. OUP Oxford, 2006.

[Koc+] Mark Koch et al. *GUPPY: Pythonic Quantum-Classical Programming*. https://popl24.sigplan.org/details/planqc-2024-papers/8/GUPPY-Pythonic-Quantum-Classical-Programming.

[Kot+23] Suhas Kotha et al. 'Provably bounding neural network preimages'. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 80270–80290.

[Kri+20] Hlér Kristjánsson et al. 'Resource theories of communication'. In: *New Journal of Physics* 22.7 (2020), p. 073014. DOI: 10.1088/1367-2630/ab8ef7.

[KSS09] Uday P. Khedker, Amitabha Sanyal and Bageshree Sathe. *Data Flow Analysis: Theory and Practice*. 1st. CRC Press, 2009. DOI: 10.1201/9780849332517. URL: https://doi.org/10.1201/9780849332517.

[KSV02] A. Yu. Kitaev, A. H. Shen and M. N. Vyalyi. *Classical and Quantum Computation*. American Mathematical Society, 2002.

[Kud22] Yu. L. Kudryashov. 'Dilatations of Linear Operators'. In: *Journal of Mathematical Sciences* 265.6 (2022), pp. 1411–1473. DOI: 10.1007/s10958-022-06093-3. URL: https://doi.org/10.1007/s10958-022-06093-3.

[Lam+08] Michael Lampis et al. 'Quantum Data and Control Made Easier'. In: *Electron. Notes Theor. Comput. Sci.* 210 (July 2008), pp. 85–105. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2008.04.020. URL: https://doi.org/10.1016/j.entcs.2008.04.020.

[LBZ21] Ji Liu, Luciano Bello and Huiyang Zhou. 'Relaxed Peephole Optimization: A Novel Compiler Optimization for Quantum Circuits'. In: *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 2021, pp. 301–314. DOI: 10.1109/CGO51591.2021.9370310.

[LCB10] Yann LeCun, Corinna Cortes and CJ Burges. 'MNIST handwritten digit database'. In: (2010). URL: http://yann.lecun.com/exdb/mnist.

[LDD20] Sirui Lu, Lu-Ming Duan and Dong-Ling Deng. 'Quantum adversarial machine learning'. In: *Physical Review Research* 2.3 (2020), p. 033212.

[Lin22] Lin Lin. 'Lecture Notes on Quantum Algorithms for Scientific Computation'. In: *CoRR* abs/2201.08309 (2022). arXiv: 2201.08309. URL: https://arxiv.org/abs/2201.08309.

[Liu+21] Changliu Liu et al. 'Algorithms for verifying deep neural networks'. In: *Foundations and Trends® in Optimization* 4.3-4 (2021), pp. 244–404.

[LP98] Noah Linden and Sandu Popescu. 'The Halting problem for quantum computers'. In: *arXiv preprint quant-ph/9806054* (June 1998). arXiv: quant-ph/9806054.

[Mar+23] Luca Marzari et al. 'The #DNN-Verification problem: Counting Unsafe Inputs for Deep Neural Networks'. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2023, pp. 217–224.

[Mar+24] Luca Marzari et al. 'Enumerating Safe Regions in Deep Neural Networks with Provable Probabilistic Guarantees'. In: *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*. Ed. by Michael J. Wooldridge, Jennifer G. Dy and Sriraam Natarajan. AAAI Press, 2024, pp. 21387–21394. DOI: 10.1609/AAAI.V38I19.30134. URL: https://doi.org/10.1609/aaai.v38i19.30134.

[Mas24] Isabella Mastroeni. 'Abstract domain adequacy'. In: *Int. J. Softw. Tools Technol. Transf.* 26.6 (2024), pp. 747–765. DOI: 10.1007/S10009-024-00774-X. URL: https://doi.org/10.1007/s10009-024-00774-x.

[Mat+21] Denny Mattern et al. 'Variational quanvolutional neural networks with enhanced image encoding'. In: *arXiv preprint arXiv:2106.07327* (2021).

[Mat+22] Andrea Matic et al. 'Quantum-classical convolutional neural networks in radiological image classification'. In: *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*. 2022, pp. 56–66. DOI: 10.1109/QCE53715.2022.00024.

[Mat13] A. Mattuck. *Introduction to Analysis*. CreateSpace Independent Publishing Platform, 2013. ISBN: 9781484814116. URL: https://books.google.it/books?id=mYppngEACAAJ.

[Min67] Marvin L. Minsky. *Computation: finite and infinite machines*. Englewood Cliffs, N.J. : Prentice-Hall, 1967.

[Mit+18] K. Mitarai et al. 'Quantum circuit learning'. In: *Phys. Rev. A* 98 (3 Sept. 2018), p. 032309. DOI: 10.1103/PhysRevA.98.032309. URL: https://link.aps.org/doi/10.1103/PhysRevA.98.032309.

[MKC09] Ramon E. Moore, R. Baker Kearfott and Michael J. Cloud. *Introduction to Interval Analysis*. Society for Industrial and Applied Mathematics, 2009. DOI: 10.1137/1.9780898717716.

[MMF25] Luca Marzari, Isabella Mastroeni and Alessandro Farinelli. 'Advancing Neural Network Verification through Hierarchical Safety Abstract Interpretation'. In: *arXiv preprint arXiv:2505.05235* (2025).

[MO05] Takayuki Miyadera and Masanori Ohya. 'On Halting Process of Quantum Turing Machine'. In: *Open Systems & Information Dynamics* 12.3 (2005), pp. 261–264. ISSN: 1573-1324. DOI: 10.1007/s11080-005-0923-2. URL: https://doi.org/10.1007/s11080-005-0923-2.

[Mot+04] Mikko Mottonen et al. 'Transformation of quantum states using uniformly controlled rotations'. In: *arXiv preprint quant-ph/0407010* (2004).

[Mun24] Nidhi Munikote. *Comparing Quantum Encoding Techniques*. 2024. arXiv: 2410.09121 [quant-ph]. URL: https://arxiv.org/abs/2410.09121.

[Mye97] John M. Myers. 'Can a Universal Quantum Computer Be Fully Quantum?' In: *Phys. Rev. Lett.* 78 (9 Mar. 1997), pp. 1823–1824. DOI: 10.1103/PhysRevLett.78.1823. URL: https://link.aps.org/doi/10.1103/PhysRevLett.78.1823.

[Nas+25] Maciej Nasinski et al. *gridify: Enrich Figures and Tables with Custom Headers and Footers and More*. R package version 0.7.5. 2025. URL: https://pharmaverse.github.io/gridify/.

[NC10] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[NNH99] Flemming Nielson, Hanne Riis Nielson and Chris Hankin. *Principles of program analysis*. Springer, 1999. ISBN: 978-3-540-65410-0. DOI: 10.1007/978-3-662-03811-6. URL: https://doi.org/10.1007/978-3-662-03811-6.

[OHe19] Peter W. O'Hearn. 'Incorrectness logic'. In: *Proc. ACM Program. Lang.* 4.POPL (Dec. 2019). DOI: 10.1145/3371078. URL: https://doi.org/10.1145/3371078.

[Par+21] Anouk Paradis et al. 'Unqomp: synthesizing uncomputation in Quantum circuits'. In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. PLDI 2021. Virtual, Canada: Association for Computing Machinery, 2021, pp. 222–236. ISBN: 9781450383912. DOI: 10.1145/3453483.3454040. URL: https://doi.org/10.1145/3453483.3454040.

[PB00] Arun Kumar Pati and Samuel L. Braunstein. 'Impossibility of deleting an unknown quantum state'. In: *Nature* 404.6774 (2000), pp. 164–165. ISSN: 1476-4687. DOI: 10.1038/404130b0. URL: https://doi.org/10.1038/404130b0.

[PBG22] Anurudh Peduri, Siddharth Bhat and Tobias Grosser. 'QSSA: an SSA-based IR for Quantum computing'. In: *Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction*. CC 2022. Seoul, South Korea: Association for Computing Machinery, 2022, pp. 2–14. ISBN: 9781450391832. DOI: 10.1145/3497776.3517772. URL: https://doi.org/10.1145/3497776.3517772.

[PBV24] Anouk Paradis, Benjamin Bichsel and Martin T. Vechev. 'Reqomp: Space-constrained Uncomputation for Quantum Circuits'. In: *Quantum* 8 (2024), p. 1258. DOI: 10.22331/Q-2024-02-19-1258. URL: https://doi.org/10.22331/q-2024-02-19-1258.

[Pea05] Karl Pearson. 'The Problem of the Random Walk'. In: *Nature* 72.1865 (1905), pp. 294–294. ISSN: 1476-4687. DOI: 10.1038/072294b0. URL: https://doi.org/10.1038/072294b0.

[Ped+11] F. Pedregosa et al. 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[Per07] Simon Perdrix. 'Quantum Patterns and Types for Entanglement and Separability'. In: *Electronic Notes in Theoretical Computer Science* 170 (2007). Proceedings of the 3rd International Workshop on Quantum Programming Languages (QPL 2005), pp. 125–138. ISSN: 1571-0661. DOI: https://doi.org/10.1016/j.entcs.2006.12.015. URL: https://www.sciencedirect.com/science/article/pii/S157106610700059X.

[Per08a] S Perdrix. 'A hierarchy of quantum semantics'. In: *Electronic Notes in Theoretical Computer Science* 192.3 (2008), pp. 71–83.

[Per08b] Simon Perdrix. 'Quantum Entanglement Analysis Based on Abstract Interpretation'. In: *Static Analysis, 15th International Symposium, SAS 2008, Valencia, Spain, July 16-18, 2008. Proceedings*. Ed. by María Alpuente and Germán Vidal. Vol. 5079. Lecture Notes in Computer Science. Springer, 2008, pp. 270–282. DOI: 10.1007/978-3-540-69166-2"˙18. URL: https://doi.org/10.1007/978-3-540-69166-2%5C˙18.

[PP23] Matteo Paltenghi and Michael Pradel. 'LintQ: A Static Analysis Framework for Qiskit Quantum Programs'. In: *arXiv preprint arXiv:2310.00718* (2023).

[PRS17] Alex Parent, Martin Roetteler and Krysta M. Svore. 'REVS: A Tool for Space-Optimized Reversible Circuit Synthesis'. In: *Reversible Computation*. Ed. by

Iain Phillips and Hafizur Rahaman. Cham: Springer International Publishing, 2017, pp. 90–101. ISBN: 978-3-319-59936-6.

[PRZ17] Jennifer Paykin, Robert Rand and Steve Zdancewic. 'QWIRE: a core language for quantum circuits'. In: *SIGPLAN Not.* 52.1 (Jan. 2017), pp. 846–858. ISSN: 0362-1340. DOI: 10.1145/3093333.3009894. URL: https://doi.org/10.1145/3093333.3009894.

[PT10] Luca Pulina and Armando Tacchella. 'An Abstraction-Refinement Approach to Verification of Artificial Neural Networks'. In: *Computer Aided Verification*. Ed. by Tayssir Touili, Byron Cook and Paul Jackson. Vol. 6174. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 243–257. DOI: 10.1007/978-3-642-14295-6˙24. URL: https://doi.org/10.1007/978-3-642-14295-6˙24.

[PYW22] 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*. PLDI 2022. San Diego, CA, USA: Association for Computing Machinery, 2022, pp. 657–670. ISBN: 9781450392655. DOI: 10.1145/3519939.3523713. URL: https://doi.org/10.1145/3519939.3523713.

[Ran+18] Robert Rand et al. 'ReQWIRE: Reasoning about Reversible Quantum Circuits'. In: EPTCS 287 (2018). Ed. by Peter Selinger and Giulio Chiribella, pp. 299–312. DOI: 10.4204/EPTCS.287.17. URL: https://doi.org/10.4204/EPTCS.287.17.

[Ran+20] Robert Rand et al. 'Gottesman Types for Quantum Programs'. In: EPTCS 340 (2020). Ed. by Benoît Valiron et al., pp. 279–290. DOI: 10.4204/EPTCS.340.14. URL: https://doi.org/10.4204/EPTCS.340.14.

[Ran+21] Robert Rand et al. 'Extending gottesman types beyond the clifford group'. In: *The Second International Workshop on Programming Languages for Quantum Computing (PLanQC 2021)*. 2021.

[RBW25] Damian Rovara, Lukas Burgholzer and Robert Wille. 'A Framework for Debugging Quantum Programs'. In: *2025 IEEE International Conference on Quantum Software (QSW)*. 2025, pp. 130–136. DOI: 10.1109/QSW67625.2025.00024.

[RC91] Horn Roger and R Johnson Charles. *Topics in matrix analysis*. 1991.

[RD24] M. Rath and H. Date. 'Quantum data encoding: a comparative analysis of classical-to-quantum mapping techniques and their impact on machine learning accuracy'. In: *EPJ Quantum Technology* 11 (2024), p. 72. DOI: 10.1140/

epjqt/s40507-024-00285-3. URL: https://doi.org/10.1140/epjqt/s40507-024-00285-3.

[Rei+25] Israel Reichental et al. 'Scalable Memory Recycling for Large Quantum Programs'. In: *arXiv preprint arXiv:2503.00822* (2025).

[Ric65] P Richard. 'Feynman AR Hibbs,"' in: *Quantum Mechanics and Path Integrals," ISBN-10* 70206503 (1965).

[RMN25] Aske Nord Raahauge, Martin Bom Marchioro and Rasmus Ross Nylandsted. 'Approximating Entanglement Based on Abstract Interpretation'. In: *arXiv preprint arXiv:2508.10056* (2025).

[Rud87] Walter Rudin. *Real and complex analysis, 3rd ed.* USA: McGraw-Hill, Inc., 1987. ISBN: 0070542341.

[RY20] Xavier Rival and Kwangkeun Yi. *Introduction to Static Analysis: An Abstract Interpretation Perspective*. en. MIT Press, Feb. 2020. ISBN: 978-0-262-35665-7.

[Sch+20] Maria Schuld et al. 'Circuit-centric quantum classifiers'. In: *Phys. Rev. A* 101 (3 Mar. 2020), p. 032308. DOI: 10.1103/PhysRevA.101.032308. URL: https://link.aps.org/doi/10.1103/PhysRevA.101.032308.

[Sei+23] Raphael Seidel et al. 'Uncomputation in the Qrisp High-Level Quantum Programming Framework'. In: *Reversible Computation - 15th International Conference, RC 2023, Giessen, Germany, July 18-19, 2023, Proceedings*. Ed. by Martin Kutrib and Uwe Meyer. Vol. 13960. Lecture Notes in Computer Science. Springer, 2023, pp. 150–165. DOI: 10.1007/978-3-031-38100-3"˙11. URL: https://doi.org/10.1007/978-3-031-38100-3%5C˙11.

[Sei+24] Raphael Seidel et al. *Qrisp: A Framework for Compilable High-Level Programming of Gate-Based Quantum Computers*. 2024. DOI: 10.48550/ARXIV.2406.14792. arXiv: 2406.14792. URL: https://doi.org/10.48550/arXiv.2406.14792.

[Sel04] Peter Selinger. 'Towards a quantum programming language'. In: *Mathematical Structures in Computer Science* 14.4 (2004), pp. 527–586. DOI: 10.1017/S0960129504004256.

[Shc13] V. S. Shchesnovich. *The second quantization method for indistinguishable particles (Lecture Notes in Physics, UFABC 2010)*. 2013. arXiv: 1308.3275 [cond-mat.quant-gas]. URL: https://arxiv.org/abs/1308.3275.

[Shi+25] Jinjing Shi et al. 'QuanTest: Entanglement-Guided Testing of Quantum Neural Network Systems'. In: *ACM Trans. Softw. Eng. Methodol.* 34.2 (Jan. 2025). ISSN: 1049-331X. DOI: 10.1145/3688840. URL: https://doi.org/10.1145/3688840.

[Shi02] Yu Shi. 'Remarks on universal quantum computer'. In: *Physics Letters A* 293.5 (2002), pp. 277–282. ISSN: 0375-9601. DOI: https://doi.org/10.1016/

S0375-9601(02)00015-4. URL: https://www.sciencedirect.com/science/article/pii/S0375960102000154.

[Sin+19] Gagandeep Singh et al. 'An abstract domain for certifying neural networks'. In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), pp. 1–30.

[Siv+20] Seyon Sivarajah et al. 't—ket⟩: a retargetable compiler for NISQ devices'. In: *Quantum Science and Technology* 6.1 (Nov. 2020), p. 014003. DOI: 10.1088/2058-9565/ab8e92. URL: https://doi.org/10.1088/2058-9565/ab8e92.

[Son08] Daegene Song. 'Unsolvability of the halting problem in quantum dynamics'. In: *International Journal of Theoretical Physics* 47 (2008), pp. 1785–1791.

[Spi08] Michael Spivak. *Calculus*. Fourth. Publish or Perish, 2008.

[SSM21] Maria Schuld, Ryan Sweke and Johannes Jakob Meyer. 'Effect of data encoding on the expressive power of variational quantum-machine-learning models'. In: *Phys. Rev. A* 103 (3 Mar. 2021), p. 032430. DOI: 10.1103/PhysRevA.103.032430. URL: https://link.aps.org/doi/10.1103/PhysRevA.103.032430.

[SSP15] Maria Schuld, Ilya Sinayskiy and Francesco Petruccione. 'An introduction to quantum machine learning'. In: *Contemporary Physics* 56.2 (2015), pp. 172–185.

[Svo+18] Krysta Svore et al. 'Q#: Enabling Scalable Quantum Computing and Development with a High-Level DSL'. In: *Proceedings of the Real World Domain Specific Languages Workshop 2018*. RWDSL2018. Vienna, Austria: Association for Computing Machinery, 2018. ISBN: 9781450363556. DOI: 10.1145/3183895.3183901. URL: https://doi.org/10.1145/3183895.3183901.

[SVV18] Amr Sabry, Benoît Valiron and Juliana Kaizer Vizzotto. 'From Symmetric Pattern-Matching to Quantum Control'. In: *Foundations of Software Science and Computation Structures*. Springer International Publishing, 2018, pp. 348–364. ISBN: 978-3-319-89366-2. DOI: 10.1007/978-3-319-89366-2˙19.

[SWH12] Helmut Seidl, Reinhard Wilhelm and Sebastian Hack. *Compiler Design - Analysis and Transformation*. Springer, 2012. ISBN: 978-3-642-17547-3. DOI: 10.1007/978-3-642-17548-0. URL: https://doi.org/10.1007/978-3-642-17548-0.

[Sze+13] Christian Szegedy et al. 'Intriguing properties of neural networks'. In: *arXiv preprint arXiv:1312.6199* (2013).

[Tiw+25] Ashutosh Tiwari et al. 'Uncomputing Ancilla Qubits in Quantum Circuits'. In: *2025 International Conference on Quantum Communications, Networking, and Computing (QCNC)*. 2025, pp. 379–387. DOI: 10.1109/QCNC64685.2025.00065.

[Val22] Benoît Valiron. 'Semantics of quantum programming languages: Classical control, quantum control'. In: *J. Log. Algebraic Methods Program.* 128 (2022), p. 100790. DOI: 10.1016/J.JLAMP.2022.100790. URL: https://doi.org/10.1016/j.jlamp.2022.100790.

[Van] Paolo Vanini. *Analysis, Probability,Functional Analysis XII*. en.

[VG25] Francisca Vasconcelos and András Gilyén. 'Methods for Reducing Ancilla-Overhead in Block Encodings'. In: *arXiv preprint arXiv:2507.07900* (2025).

[Vid00] Guifré Vidal. 'Entanglement monotones'. In: *Journal of Modern Optics* 47.2-3 (2000), pp. 355–376. DOI: 10.1080/09500340008244048. eprint: https://www.tandfonline.com/doi/pdf/10.1080/09500340008244048. URL: https://www.tandfonline.com/doi/abs/10.1080/09500340008244048.

[Voi+23] Finn Voichick et al. 'Qunity: A Unified Language for Quantum and Classical Computing'. In: *Proc. ACM Program. Lang.* 7.POPL (Jan. 2023). DOI: 10.1145/3571225. URL: https://doi.org/10.1145/3571225.

[Wan+18] Shiqi Wang et al. 'Formal security analysis of neural networks using symbolic intervals'. In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 1599–1614.

[Wan+21] Shiqi Wang et al. 'Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification'. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 29909–29921.

[War42] Morgan Ward. 'The Closure Operators of a Lattice'. In: *Annals of Mathematics* 43.2 (1942), pp. 191–196. ISSN: 0003486X, 19398980. DOI: 10.2307/1968865. URL: http://www.jstor.org/stable/1968865 (visited on 12/11/2024).

[Web+21] Maurice Weber et al. 'Optimal provable robustness of quantum classification via quantum hypothesis testing'. In: *npj Quantum Information* 7.1 (2021), p. 76. DOI: 10.1038/s41534-021-00410-5. URL: https://doi.org/10.1038/s41534-021-00410-5.

[Wei+25] Tianhao Wei et al. 'Modelverification. jl: a comprehensive toolbox for formally verifying deep neural networks'. In: *Proceedings of the 37th International Conference on Computer Aided Verification*. 2025.

[Wet20] John van de Wetering. *ZX-calculus for the working quantum computer scientist*. 2020. arXiv: 2012.13966 [quant-ph]. URL: https://arxiv.org/abs/2012.13966.

[Win93] Glynn Winskel. *The formal semantics of programming languages: an introduction*. MIT press, 1993.

[Wri+24] Christopher John Wright et al. 'Quff: A Dynamically Typed Hybrid Quantum-Classical Programming Language'. In: *Proceedings of the 21st ACM SIGPLAN*

International Conference on Managed Programming Languages and Runtimes. MPLR 2024. Vienna, Austria: Association for Computing Machinery, 2024, pp. 65–81. ISBN: 9798400711183. DOI: 10.1145/3679007.3685063. URL: https://doi.org/10.1145/3679007.3685063.

[WTD24] Maximilian Wendlinger, Kilian Tscharke and Pascal Debus. 'A comparative analysis of adversarial robustness for quantum and classical machine learning models'. In: *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Vol. 1. IEEE. 2024, pp. 1447–1457.

[WZ82] W. K. Wootters and W. H. Zurek. 'A single quantum cannot be cloned'. In: *Nature* 299.5886 (1982), pp. 802–803. ISSN: 1476-4687. DOI: 10.1038/299802a0. URL: https://doi.org/10.1038/299802a0.

[Xu+20] Kaidi Xu et al. 'Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers'. In: *arXiv preprint arXiv:2011.13824* (2020).

[XZ23] Shangzhou Xia and Jianjun Zhao. 'Static Entanglement Analysis of Quantum Programs'. In: *4th IEEE/ACM International Workshop on Quantum Software Engineering, Q-SE@ICSE 2023, Melbourne, Australia, May 17, 2023*. IEEE, 2023, pp. 42–49. DOI: 10.1109/Q-SE59154.2023.00013. URL: https://doi.org/10.1109/Q-SE59154.2023.00013.

[Yam+14] Fabian Yamaguchi et al. 'Modeling and Discovering Vulnerabilities with Code Property Graphs'. In: *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. IEEE Computer Society, 2014, pp. 590–604. DOI: 10.1109/SP.2014.44. URL: https://doi.org/10.1109/SP.2014.44.

[YF10] Mingsheng Ying and Yuan Feng. 'Quantum loop programs'. In: *Acta Informatica* 47.4 (2010), pp. 221–250.

[Yin12] Mingsheng Ying. 'Floyd–hoare logic for quantum programs'. In: *ACM Trans. Program. Lang. Syst.* 33.6 (Jan. 2012). ISSN: 0164-0925. DOI: 10.1145/2049706.2049708. URL: https://doi.org/10.1145/2049706.2049708.

[Yin14] Mingsheng Ying. 'Quantum recursion and second quantisation'. In: *arXiv preprint arXiv:1405.4443* (2014).

[Yin16] Mingsheng Ying. *Foundations of quantum programming*. Morgan Kaufmann, 2016.

[YMC22] Charles Yuan, Christopher McNally and Michael Carbin. 'Twist: sound reasoning for purity and entanglement in Quantum programs'. In: *Proc. ACM Program. Lang.* 6.POPL (Jan. 2022). DOI: 10.1145/3498691. URL: https://doi.org/10.1145/3498691.

[YP21] 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. Virtual, Canada: Association for Computing Machinery, 2021, pp. 542–558. ISBN: 9781450383912. DOI: 10.1145/3453483.3454061. URL: https://doi.org/10.1145/3453483.3454061.

[YPR25] Nengkun Yu, Jens Palsberg and Thomas Reps. 'SAQR-QC: A Logic for Scalable but Approximate Quantitative Reasoning about Quantum Circuits'. In: *CoRR* abs/2507.13635 (2025). DOI: 10.48550/ARXIV.2507.13635. arXiv: 2507.13635. URL: https://doi.org/10.48550/arXiv.2507.13635.

[YVC24] Charles Yuan, Agnes Villanyi and Michael Carbin. 'Quantum Control Machine: The Limits of Control Flow in Quantum Programming'. In: *Proc. ACM Program. Lang.* 8.OOPSLA1 (Apr. 2024). DOI: 10.1145/3649811. URL: https://doi.org/10.1145/3649811.

[YYF12] Mingsheng Ying, Nengkun Yu and Yuan Feng. 'Defining Quantum Control Flow'. In: *CoRR* abs/1209.4379 (2012). arXiv: 1209.4379. URL: http://arxiv.org/abs/1209.4379.

[YYF14] Mingsheng Ying, Nengkun Yu and Yuan Feng. 'Alternation in Quantum Programming: From Superposition of Data to Superposition of Programs'. In: *CoRR* abs/1402.5172 (2014). arXiv: 1402.5172. URL: http://arxiv.org/abs/1402.5172.

[Zha+18] Huan Zhang et al. 'Efficient neural network robustness certification with general activation functions'. In: *Advances in neural information processing systems* 31 (2018).

[Zha+21] Pengzhan Zhao et al. 'Bugs4Q: A Benchmark of Real Bugs for Quantum Programs'. In: *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 2021, pp. 1373–1376. DOI: 10.1109/ASE51524.2021.9678908. URL: https://doi.org/10.1109/ASE51524.2021.9678908.

[Zha+23] Pengzhan Zhao et al. 'QChecker: Detecting Bugs in Quantum Programs via Static Analysis'. In: *4th IEEE/ACM International Workshop on Quantum Software Engineering, Q-SE@ICSE 2023, Melbourne, Australia, May 17, 2023*. IEEE, 2023, pp. 50–57. DOI: 10.1109/Q-SE59154.2023.00014. URL: https://doi.org/10.1109/Q-SE59154.2023.00014.

[ZWK24] Xiyue Zhang, Benjie Wang and Marta Kwiatkowska. 'Provable preimage under-approximation for neural networks'. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2024, pp. 3–23.

[ZY25] Zhicheng Zhang and Mingsheng Ying. 'Quantum Register Machine: Efficient Implementation of Quantum Recursive Programs'. In: *Proc. ACM Program. Lang.* 9.PLDI (June 2025). DOI: 10.1145/3729283. URL: https://doi.org/10.1145/3729283.

[ZZM21] Pengzhan Zhao, Jianjun Zhao and Lei Ma. 'Identifying Bug Patterns in Quantum Programs'. In: *2nd IEEE/ACM International Workshop on Quantum Software Engineering, Q-SE@ICSE 2021, Madrid, Spain, June 1-2, 2021*. IEEE, 2021, pp. 16–21. DOI: 10.1109/Q-SE52541.2021.00011. URL: https://doi.org/10.1109/Q-SE52541.2021.00011.