# ALGEBRAIC AND LOGICAL METHODS IN QUANTUM COMPUTATION

by

Neil J. Ross

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
August 2015

*À mes maîtres.*

# Table of Contents

# List of Figures

# Abstract

This thesis contains contributions to the theory of quantum computation.

We first define a new method to efficiently approximate special unitary operators. Specifically, given a special unitary $U$ and a precision $\varepsilon > 0$, we show how to efficiently find a sequence of Clifford+$V$ or Clifford+$T$ operators whose product approximates $U$ up to $\varepsilon$ in the operator norm. In the general case, the length of the approximating sequence is asymptotically optimal. If the unitary to approximate is diagonal then our method is optimal: it yields the shortest sequence approximating $U$ up to $\varepsilon$.

Next, we introduce a mathematical formalization of a fragment of the Quipper quantum programming language. We define a typed lambda calculus called Proto-Quipper which formalizes a restricted but expressive fragment of Quipper. The type system of Proto-Quipper is based on intuitionistic linear logic and prohibits the duplication of quantum data, in accordance with the no-cloning property of quantum computation. We prove that Proto-Quipper is type-safe in the sense that it enjoys the subject reduction and progress properties.

# List of Abbreviations and Symbols Used

| | |
|---|---|
| $\mathbb{N}$ | The natural numbers. |
| $\mathbb{Z}$ | The integers. |
| $\mathbb{Q}$ | The rational numbers. |
| $\mathbb{R}$ | The real numbers. |
| $\mathbb{C}$ | The complex numbers. |
| $R^{\times}$ | The group of units of the ring $R$. |
| $\lVert . \rVert$ | The norm of a scalar, vector, or matrix. |
| $\otimes$ | The tensor product. |
| $\mathcal{M}_{m,n}(R)$ | The set of $m$ by $n$ matrices over the ring $R$. |
| $\mathrm{U}(n)$ | The unitary group of order $n$. |
| $\mathrm{SU}(n)$ | The special unitary group of order $n$. |
| $U^{-1}$ | The inverse of the matrix $U$. |
| $U^{\dagger}$ | The conjugate transpose of the matrix $U$. |
| $\det(U)$ | The determinant of the matrix $U$. |
| $\mathrm{tr}(U)$ | The trace of the matrix $U$. |
| $K(\alpha)$ | The extension of the field $K$ by $\alpha$. |
| $R[\alpha]$ | The extension of the ring $R$ by $\alpha$. |
| $\mathcal{O}_K$ | The ring of integers of the field $K$. |
| $(.)^{\bullet}$ | The bullet automorphism of $\mathbb{Z}[\omega]$. |
| $\equiv$ | Modular congruence. |
| $\mid$ | Divisibility. |
| $\mathcal{N}_R(.)$ | The norm for the ring of integers $R$. |
| $a\{y/x\}$ | The renaming of $x$ by $y$ in $a$. |
| $\mathrm{FV}(a)$ | The free variables of $a$. |
| $a[b/x]$ | The substitution of $b$ for $x$ in $a$. |
| $\to, \twoheadrightarrow$ | The one-step and multi-step $\beta$-reduction. |
| $=_{\alpha}$ | The $\alpha$-equivalence. |
| $\equiv_{\beta}$ | The $\beta$-equivalence. |

| | |
|---|---|
| $<:$ | The subtyping relation. |
| $[Q, L, a]$ | A closure of the quantum lambda calculus. |
| $\mathrm{up}(A)$ | The uprightness of the set $A$. |
| $\mathrm{BBox}(A)$ | The bounding box of the set $A$. |
| $\mathrm{area}(A)$ | The area of the set $A$. |
| $\mathrm{Grid}(A)$ | The grid for the set $A$. |
| $O(.)$ | The big-$O$ notation. |
| $\mathtt{Skew}(D)$ | The skew of the ellipse $D$. |
| $\mathtt{Bias}(D, \Delta)$ | The bias of the state $(D, \Delta)$. |
| $\sinh_\lambda(.)$ | The hyperbolic sine in base $\lambda$. |
| $\cosh_\lambda(.)$ | The hyperbolic cosine in base $\lambda$. |
| $\mathcal{R}_\varepsilon$ | The $\varepsilon$-region. |
| $\overline{\mathcal{D}}$ | The closed unit disk. |
| $\mathrm{Re}(a)$ | The real part of the complex number $a$. |
| $\mathcal{P}_f(X)$ | The set of finite subsets of the set $X$. |
| $\mathrm{FQ}(a)$ | The free quantum variables of a term $a$. |
| $\mathrm{Bij}_f(X)$ | The set of finite bijections on the set $X$. |
| $\mathtt{Spec}_X(T)$ | An $X$-specimen for $T$. |
| $[C, a]$ | A closure of Proto-Quipper. |
| $\lfloor . \rfloor$ | The floor function. |
| $\uplus$ | The disjoint union. |

## Acknowledgements

I would like to express my profound gratitude to the people who, in one way or another, have contributed to the writing of this thesis.

I want to thank my supervisor Peter Selinger, for his insight, his guidance, and, most importantly, his contagious passion for mathematics. Special thanks are due to the members of my supervisory committee, Dorette Pronk and Richard Wood, for accepting to read a thesis which, alas, contains so few arrows. I am very grateful to Prakash Panangaden for accepting to be my external examiner.

The staff of the Department of Mathematics and Statistics of Dalhousie University have played a large part in making my stay in Halifax enjoyable. I am very appreciative of their hard work.

I am greatly indebted to the many teachers I had the chance to learn from during the past ten years. In particular, I want to acknowledge the profound influence of Susana Berestovoy, Julien Dutant, Vincent Homer, Jean-Baptiste Joinet, and Damiano Mazza.

I want to thank the researchers I had the pleasure to collaborate with, notably D. Scott Alexander, Henri Chataing, Alexander S. Green, Peter LeFanu Lumsdaine, Jonathan M. Smith, and Benoît Valiron.

My fellow students have always provided inspiration, support, and laughter. I especially want to thank Abdullah Al-Shaghay, Ali Alilooee, Chloé Berruyer, Samir Blakaj, Méven Cadet, Antonio Chavez, Hoda Chuangpishit, Hugo Férée, Florent Franchette, Brett Giles, Giulio Guerrieri, François Guignot, Zhenyu Victor Guo, D. Leigh Herman, Ben Hersey, Joey Mingrone, Lucas Mol, Alberto Naibo, Mattia Petrolo, Francisco Rios, Kira Scheibelhut, Matthew Stephen, Aurélien Tonneau, Antonio Vargas, Kim Whoriskey, Bian Xiaoning, Amelia Yzaguirre, and Kevin Zatloukal.

Finally, I want to thank my friends and loved ones, far and near. My brother and my sister. My parents, for their unwavering support. And Kira, for her kindness.

# Chapter 1

# Introduction

*Quantum computation*, introduced in the early 1980s by Feynman [18], is a paradigm for computation based on the laws of quantum physics. The interest in quantum computation lies in the fact that *quantum computers* can solve certain problems more efficiently than their *classical* counterparts. Most famously, Shor showed in 1994 that integers can be factored in polynomial time on a quantum computer [62]. This is in striking contrast with the exponential running time of the best known classical methods. Since then, many algorithms leveraging the power of quantum computers have been introduced (e.g., [29], [48], [30]). This promised increase in efficiency has provided great incentive to solve the theoretical and practical challenges associated with building quantum computers.

The fundamental unit of quantum computation is the *quantum bit* or *qubit.* The *state* of a qubit is described by a unit vector in the two-dimensional Hilbert space $\mathbb{C}^2$. A system of $n$ qubits is similarly described by a unit vector in the Hilbert space

$$\mathbb{C}^{2^n} = \underbrace{\mathbb{C}^2 \otimes \ldots \otimes \mathbb{C}^2}_{n}.$$

A computation is performed by acting on the state of a system of qubits. This can be done in two ways. One can either apply a *unitary transformation* to the state or one can *measure* some of the qubits making up the system. A quantum algorithm describes a sequence of unitary operations to be performed on the state of a system of qubits, usually followed by a single final measurement, or in some cases, measurements throughout the computation. It is customary to represent a sequence of unitary operations in the form of a *quantum circuit*, which is built up from a basic set of unitaries, called *gates*, using composition and tensor product. An important peculiarity of quantum computation is that the state of a quantum system cannot in general be *duplicated.* This is the so-called *no-cloning* property. This contrasts with the situation in classical computation, where the state of a bit can be freely copied.

Quantum algorithms are run on a physical device. Just as in classical computing, there is a chain of successive translations, starting from a mathematical description of an algorithm in a research paper. The abstract algorithm is first implemented in a programming language, which is then turned into a quantum circuit. This first circuit is then rewritten using a finite set of basic unitaries available on the hardware. Finally, this second circuit is rewritten according to some error correcting scheme, which redundantly encodes the circuit to make it robust to some of the errors that are bound to occur on the physical machine. At this point, the abstract description of the algorithm can be mapped to a physical system for the computation to be effectively performed.

All of the above-mentioned steps in the execution of a quantum algorithm raise interesting mathematical questions. For many years the problems at the top of this translation chain, those closer to the abstract description of the algorithm, were either overlooked or considered adequately solved. This was justified by the fact that the challenges of building a reliable physical quantum computer were so far from being met that the higher-level problems seemed somewhat irrelevant. However, as the prospect of usable quantum computers draws nearer, the need to develop tools to define proper solutions to these problems has become more pressing.

In this thesis, we contribute to two of the mathematical problems that arise in the higher level of this execution phase. We first develop methods to decompose unitaries into certain basic sets of unitaries. Secondly, we introduce a lambda calculus which serves as a foundation for the *Quipper* quantum programming language. We now briefly outline each of these contributions.

## 1.1 Approximate synthesis

The *unitary group of order 2*, denoted $\mathrm{U}(2)$, is the group of $2 \times 2$ complex unitary matrices. The *special unitary group of order 2*, denoted $\mathrm{SU}(2)$, is the subgroup of $\mathrm{U}(2)$ consisting of unitary matrices of determinant 1.

Let $S \subseteq \mathrm{U}(2)$ be a set of unitaries and $\langle S \rangle$ be the set of words over $S$. In the context of quantum computing, the elements of $S$ are called *single-qubit gates* and the elements of $\langle S \rangle$ are called *single-qubit circuits over $S$*. Matrix multiplication defines a map $\mu : \langle S \rangle \to \mathrm{U}(2)$. However, we often abuse notation and simply write $W$ for

$\mu(W)$.

We think of $S \subseteq \mathrm{U}(2)$ as the set of unitary operations that can be performed natively on a quantum computer. Because a quantum computer is a physical device, $S$ is finite and the set $\langle S \rangle$ of circuits expressible on our quantum computer is countable. In contrast, both U(2) and SU(2) are uncountable. Hence, regardless of which $S$ is chosen, there will be $U \in \mathrm{SU}(2)$ such that $U \notin \langle S \rangle$. This is a fortiori true of U(2) also. This might be problematic, as it is often desirable to have all unitaries at our disposal when writing quantum algorithms. This tension is alleviated by the fact that unitaries can be *approximated.*

**Definition 1.1.1.** The *distance* between two operators $U, W \in \mathrm{U}(2)$ is defined as

$$\|U - W\| = \sup\{||Uv - Wv|| \ ; \ v \in \mathbb{C}^2 \text{ and } ||v|| = 1\}.$$

The notion of distance introduced in Definition 1.1.1, based on the operator norm, is adopted because the physically observable difference between two unitaries $U$ and $W$ is a function of $\|U - W\|$. As a result, if $\varepsilon$ is small enough, the action of the unitaries $U$ and $W$ are observably almost indistinguishable. The finiteness of the gate set $S$ can therefore be remedied if $\langle S \rangle$ is dense in SU(2).

**Definition 1.1.2.** A set $S \subseteq \mathrm{U}(2)$ is *universal* if for any $U \in \mathrm{SU}(2)$ and any $\varepsilon > 0$, there exists $W \in \langle S \rangle$ such that $\|U - W\| < \varepsilon$.

Note that a set $S$ of gates is universal if it is dense in SU(2), rather than in U(2). This is due to the fact that since a global phase has no observable effect in quantum mechanics, we can without loss of generality focus on special unitary matrices.

If a gate set $S$ is universal, then any special unitary can be approximated up to an arbitrarily small precision by a circuit over $S$. However, the fact that $S$ is universal does not provide an efficient method which, given a special unitary $U$ and a precision $\varepsilon$, allows us to construct the approximating circuit $W$. An algorithm that performs such a task is a solution to the *approximate synthesis problem for $S$.*

**Problem 1.1.3** (Approximate synthesis problem for $S$)**.** Given a unitary $U \in \mathrm{SU}(2)$ and a precision $\varepsilon \geqslant 0$, construct a circuit $W$ over $S$ such that $\|W - U\| \leqslant \varepsilon$.

The approximate synthesis problem is important for quantum computing because it significantly impacts the resources required to run a quantum algorithm. Indeed, a quantum circuit, to be executed by a quantum computer, must first be compiled into some universal gate set. The complexity of the final physical circuit therefore crucially depends on the chosen synthesis method. In view of the considerable resources required for most quantum algorithms on interesting problem sizes, which can require upwards of 30 trillion gates [28], a universal gate set can be realistically considered for practical quantum computing only if it comes equipped with a good synthesis algorithm.

In this thesis, we are interested in Problem 1.1.3 for two universal extensions of the *Clifford group.* The Clifford group is generated by the following gates

$$\omega = e^{i\pi/4}, \quad H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad \text{and } S = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}.$$

Note that $\omega$ is a complex number, rather than a matrix. By a slight abuse of notation, we write $\omega$ to denote the unitary $\omega I$, where $I$ is the identity $2 \times 2$ matrix. The Clifford group is of great interest in quantum computation because Clifford circuits can be fault-tolerantly implemented at very low cost in most error-correcting schemes. For this reason, the Clifford group is often seen as a prime candidate for practical quantum computing. However, the Clifford group is finite and therefore not universal for quantum computing. Moreover, Gottesman and Knill showed that Clifford circuits can be efficiently simulated on a classical computer [26]. It is therefore necessary to consider universal extensions of the Clifford group.

The first extension we will consider, the *Clifford+V* gate set, arises by adding the following *$V$-gates* to the set of Clifford generators

$$V_X = \frac{1}{\sqrt{5}}\begin{bmatrix} 1 & 2i \\ 2i & 1 \end{bmatrix}, \quad V_Y = \frac{1}{\sqrt{5}}\begin{bmatrix} 1 & 2 \\ -2 & 1 \end{bmatrix}, \text{ and } V_Z = \frac{1}{\sqrt{5}}\begin{bmatrix} 1+2i & 0 \\ 0 & 1-2i \end{bmatrix}.$$

The $V$-gates were introduced in [46] and [47] and later considered in the context of approximate synthesis in [32], [7], [54], and [5].

The second extension we will consider, the *Clifford+T* gate set, arises by adding the following $T$-gate to the set of Clifford generators

$$T = \begin{bmatrix} 1 & 0 \\ 0 & \omega \end{bmatrix}$$

where $\omega = e^{i\pi/4}$ as above. The $T$ gate is the most common extension of the Clifford group considered in the literature. One reason for this is that the Clifford+$T$ gate set enjoys nice error-correction properties [50]. This gate set has often been considered in the approximate synthesis literature (see, e.g., [41], [40], [60], and [56]).

We evaluate an approximate synthesis algorithm with respect to its *time complexity* and its *circuit complexity*. The time complexity of an algorithm is the number of arithmetic operations it requires to produce an approximating circuit. The circuit complexity is the length of the produced circuit, which we identify with the number of non-Clifford gates that appear in the circuit. This is motivated by the high cost of error correction for non-Clifford gates.

Until recently, there were two main approaches to the approximate synthesis problem: the ones based on exhaustive search, like Fowler's algorithm of [20], and the ones based on geometric methods, like the Solovay-Kitaev algorithm ([37], [14], [38]). The methods based on exhaustive search achieve optimal circuit sizes but, due to their exponential runtimes, are impractical for small $\varepsilon$. In contrast, the well-known Solovay-Kitaev algorithm has polynomial runtime and achieves circuit sizes of $O(\log^c(1/\varepsilon))$, where $c > 3$. However, the information-theoretic lower bound is $O(\log(1/\varepsilon))$, so the Solovay-Kitaev algorithm leaves ample room for improvement.

In the past few years, number theoretic methods, and in particular Diophantine equations, have been used to define new synthesis algorithms. This has rejuvenated the field of quantum circuit synthesis and significant progress has been made ([42], [7], [54], [5], [41], [40], [60], [56]).

The algorithms we introduce in this thesis belong to this new kind of number-theoretic algorithms. We give algorithms for Clifford+$V$ and Clifford+$T$ circuits. Both algorithms are efficient (they run in probabilistic polynomial time) and achieve near-optimal circuit length. In certain specific cases, the algorithms are optimal. That is, the produced circuits are the shortest approximations possible. This solves a long-standing open problem, as no such efficient optimal synthesis method was previously known for any gate set.

## 1.2 The mathematical foundations of Quipper

*Quipper* is a programming language for quantum computation (see [1], [27], [28], and [55]). The Quipper language was developed in 2011–2013 in the context of a research contract for the U.S. Intelligence Advanced Research Project Activity (see [35]). As part of this project, seven non-trivial algorithms from the quantum computing literature were implemented in Quipper ([10], [2], [30], [65], [31], [53], and [48]). I participated in the development of the Quipper language and in the implementation of the Triangle Finding Algorithm [48] and the Unique Shortest Vector Algorithm [53].

An important aspect of the Quipper language is that it acts as a *circuit description language*. This means that Quipper provides a syntax in which to express quantum circuits. Quipper moreover provides the ability to treat circuits as data and to manipulate them as a whole. For example, Quipper has operators for reversing and iterating circuits, decomposing them into gate sets, etc. This circuit-as-data paradigm is remarkably useful for the programmer, as it is very close to the way in which quantum algorithms are described in the literature.

Currently, Quipper is implemented as an *embedded* language [12]. This means that Quipper can be seen as a collection of functions and data types within some pre-existing *host* language. Quipper's host language is *Haskell* [34], a strongly-typed functional programming language. An advantage of this embedded language approach is that it allows for the implementation of a large-scale system without having to first design and implement a compiler, a parser, etc. The embedded language approach also has drawbacks, however. In particular, Quipper inherits the type system of its host and while Haskell's type system provides many type-safety properties, it is not in general strong enough to ensure the full type-safety of quantum programs. In the current Quipper implementation, it is therefore the programmer's responsibility to ensure that quantum components are plugged together in physically meaningful ways. This means that certain types of programming errors will not be prevented by the compiler. In the worst case, this may lead to ill-formed output or run-time errors.

In this thesis, we introduce a quantum programming language which we call *Proto-Quipper*. It is defined as a typed lambda calculus and can be seen as a mathematical formalization of a fragment of Quipper. Proto-Quipper is meant to provide a foundation for the development of a stand-alone (i.e., non-embedded) version of Quipper. Moreover, Proto-Quipper is designed to "enforce the physics", in the sense that it detects, at compile-time, programming errors that could lead to ill-formed or undefined circuits. In particular, the no-cloning property of quantum computation is enforced.

In designing the Proto-Quipper language, our approach was to start with a limited, but still expressive, fragment of the Quipper language and make it type safe. This fragment will serve then as a robust basis for future language extensions. The idea is to eventually close the gap between Proto-Quipper and Quipper by extending Proto-Quipper with one feature at a time while retaining type safety.

Our main inspiration for the design of Proto-Quipper is the quantum lambda calculus (see [64], [61], or [58]). The quantum lambda calculus represents an ideal starting point for the design of Proto-Quipper because it is equipped with a type system tailored for quantum computation. However, the quantum lambda calculus only manipulates qubits and all quantum operations are immediately carried out on a quantum device, not stored for symbolic manipulation. We therefore extend the quantum lambda calculus with the minimal set of features that makes it Quipper-like. The current version of Proto-Quipper is designed to

- incorporate Quipper's ability to generate and act on quantum circuits, and to
- provide a linear type system to guarantee that the produced circuits are physically meaningful (in particular, properties like no-cloning are respected).

To achieve these goals, we extend the types of the quantum lambda calculus with a type $Circ(T, U)$ of circuits, and add constant terms to capture some of Quipper's circuit-level operations, like reversing. We give a formal operational semantics of Proto-Quipper in terms of a reduction relation on pairs $[C, t]$ of a term $t$ of the language and a so-called circuit state $C$. The state $C$ represents the circuit currently being built. The reduction is defined as a rewrite procedure on such pairs, with the state being affected when terms involve quantum constants.

## 1.3 Outline

The thesis can be divided into three parts, whose contents are outlined below.

- The first part of the thesis, which corresponds to chapters 2 – 4, contains background material. Chapter 2 is an exposition of the basic notions of quantum computation. Chapter 3 introduces concepts and methods from algebraic number theory. Chapter 4 presents the lambda calculus as well as the quantum lambda calculus.

- The second part of the thesis, which corresponds to chapters 5 – 7, contains algebraic contributions to quantum computation. In Chapter 5, we introduce and solve grid problems. In Chapter 6, we define an algorithm to solve the problem of approximate synthesis of special unitaries over the Clifford+$V$ gate set. In Chapter 7 we show how the methods of the previous chapter can be adapted to the Clifford+$T$ gate set.

- The third and last part of the thesis, which corresponds to chapters 8 and 9, contains logical contributions to quantum computation. In Chapter 8 we introduce the syntax, type system, and operational semantics of Proto-Quipper. In Chapter 9 we prove that Proto-Quipper enjoys the subject reduction and progress properties.

In the interest of brevity, the introductory chapters 2 – 4 contain only the material that is necessary to the subsequent chapters. In particular, most proofs are omitted. In each of these chapters, we provide references to the relevant literature.

## 1.4 Contributions

My original contributions are contained in chapters 5 – 9 of the thesis. They have appeared in several published papers. The algorithm for the Clifford+$V$ approximate synthesis of unitaries and its analysis (Section 5.1 and Chapter 6) appeared in the single-authored paper [54]. The algorithm for the Clifford+$T$ approximate synthesis of unitaries and its analysis (Section 5.2 and Chapter 7) appeared in the paper [56], co-authored with my supervisor Peter Selinger, following earlier work by Selinger [60]. The results of Section 5.2.4, providing a method to make two ellipses simultaneously upright, are my original contribution. The other results of [56] are the product of an equal collaboration. Finally, the definition of the Proto-Quipper language, as well as the proof of its type safety (Chapter 8 and 9) appeared in the report [9], co-authored with Henri Chataing and Peter Selinger. The results in these chapters are my original work; Peter Selinger's role was supervisory, and Henri Chataing was a summer intern whom I helped supervise.

# Chapter 2

# Quantum computation

In this chapter, we provide a brief introduction to quantum computation. Further details can be found in the literature. References for this material include [50] and [36]. For a concise introduction, we also refer the reader to [59] which we loosely follow here.

## 2.1 Linear Algebra

We write $\mathbb{N}$ for the semiring of non-negative integers (including 0) and $\mathbb{Z}$ for the ring of integers. The fields of rational numbers, real numbers, and complex numbers are denoted by $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$ respectively. Recall that $\mathbb{C} = \{a + bi \ ; \ a, b \in \mathbb{R}\}$ so that the complex numbers can be identified with the two-dimensional real plane $\mathbb{R}^2$. Recall moreover that if $\alpha = a + bi$ is a complex number, then its *conjugate* is $\alpha^\dagger = a - bi$.

### 2.1.1 Finite dimensional Hilbert spaces

We will be interested in complex vector spaces of the form $\mathbb{C}^n$ for some $n \in \mathbb{N}$. The integer $n$ is called the *dimension* of $\mathbb{C}^n$. The vector space $\mathbb{C}^n$ has a canonical basis, whose elements we denote by $e_i$, for $1 \leqslant i \leqslant n$. Every element $\alpha \in \mathbb{C}^n$ can be uniquely written as a linear combination

$$\alpha = a_1 e_1 + \ldots + a_n e_n \tag{2.1}$$

with $a_1, \ldots a_n \in \mathbb{C}$. We will often represent the vector $\alpha$ of (2.1) as a *column vector*

$$\alpha = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}. \tag{2.2}$$

We identify $\mathbb{C}^1$ with $\mathbb{C}$ and frequently refer to the elements of $\mathbb{C}$ as *scalars*. The vector space $\mathbb{C}^n$ is equipped with the usual operations of addition and scalar multiplication.

Given a vector $\alpha$ as in (2.2), its *dual* is the *row vector*

$$\alpha^\dagger = \begin{bmatrix} a_1^\dagger & \dots & a_n^\dagger \end{bmatrix}.$$

Note that under the identification of $\mathbb{C}^1$ and $\mathbb{C}$, no ambiguity arises from using $(-)^\dagger$ to denote the dual of a vector and the conjugate of a scalar. The *norm* of a vector $\alpha$ is defined as $||\alpha|| = \sqrt{\alpha^\dagger\alpha}$, where $\alpha^\dagger\alpha$ is obtained by matrix multiplication. A vector whose norm is 1 is called a *unit vector*. Equipped with the norm $|| - ||$, the vector space $\mathbb{C}^n$ has the structure of an $n$-dimensional *Hilbert space*.

### 2.1.2 Operators and matrices

A linear operator $\mathbb{C}^n \to \mathbb{C}^m$ can be represented by an $m \times n$ complex matrix. We write $\mathcal{M}_{m,n}(\mathbb{C})$ for the set of all $m \times n$ complex matrices and we say that $m$ and $n$ are the *dimensions* of $U \in \mathcal{M}_{m,n}(\mathbb{C})$. If $m = n$, then we simply say that $U$ has dimension $n$. We identify $\mathcal{M}_{n,1}(\mathbb{C})$ and $\mathbb{C}^n$.

We will use some well-known operations on matrices. For any $n$, the identity matrix of dimension $n$ is denoted by $I_n$, or simply by $I$ if the dimension is clear from context. If $U \in \mathcal{M}_{n,n}(\mathbb{C})$, an *inverse* of $U$, written $U^{-1}$, is a matrix such that $UU^{-1} = U^{-1}U = I$. If it exists, the inverse of a matrix is unique. We note that for any $n$, the set of invertible matrices of dimension $n$ forms a group under matrix multiplication. We denote this group by $\mathcal{M}_{n,n}(\mathbb{C})^\times$. If $U \in \mathcal{M}_{m,n}(\mathbb{C})$, the *conjugate transpose* of $U$, written $U^\dagger$, is defined by $U^\dagger_{i,j} = (U_{j,i})^\dagger$. The identification of $\mathcal{M}_{n,1}(\mathbb{C})$ and $\mathbb{C}^n$ ensures that no ambiguity arises from our use of $(-)^\dagger$ to denote the conjugate transpose of a matrix, since the conjugate transpose of a column vector is its dual.

We will also use the well-known notions of *trace* and *determinant* of a square matrix. Both are functions that assign a complex number to any complex square matrix. For $U \in \mathcal{M}_{n,n}(\mathbb{C})$, the trace of $U$ is defined as

$$\mathrm{tr}(U) = \sum_{i=1}^{n} U_{i,i}.$$

The formula to express the determinant of an arbitrary $n \times n$ matrix is somewhat cumbersome. Since we will only be considering determinants of $2 \times 2$ matrices, we give an explicit definition of the determinant in this case only. For $U \in \mathcal{M}_{2,2}(\mathbb{C})$, the

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \quad X = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \quad Y = \begin{bmatrix}0 & -i\\ i & 0\end{bmatrix} \quad Z = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$$

Figure 2.1: The Hadamard matrix $H$ and the Pauli matrices $X$, $Y$, and $Z$.

determinant of $U$ is defined as

$$\det\left(\begin{bmatrix}\alpha & \beta\\ \gamma & \delta\end{bmatrix}\right) = \alpha\delta - \beta\gamma.$$

We note that in any dimension the determinant is multiplicative and $\det(I) = 1$.

Finally, if $U \in \mathcal{M}_{n,m}(\mathbb{C})$, $\alpha \in \mathbb{C}^n$, $\lambda \in \mathbb{C}$, and $U\alpha = \lambda\alpha$, then $\alpha$ is an *eigenvector* of $U$ with *eigenvalue* $\lambda$.

### 2.1.3 Unitary, Hermitian, and positive matrices

A complex matrix $U$ is *unitary* if $U^{-1} = U^\dagger$. Unitary matrices preserve the norm of vectors in $\mathbb{C}^n$. That is, if $U$ is unitary then $||U\alpha|| = ||\alpha||$ for any vector $\alpha \in \mathbb{C}^n$. The composition of two unitary matrices is again unitary. Moreover, the identity matrix is unitary. Hence, the set $\mathrm{U}(n)$ of all unitary matrices of dimension $n$ forms a group, called the *unitary group of order n*. Since the determinant is a multiplicative function, $\mathrm{U}(n)$ has a subgroup which consists of those unitary matrices whose determinant is 1. This group is called the *special unitary group of order n* and is denoted by $\mathrm{SU}(n)$. We thus have the inclusions

$$\mathrm{SU}(n) \subseteq \mathrm{U}(n) \subseteq \mathcal{M}_{n,n}(\mathbb{C})^\times.$$

Examples of useful unitary matrices are provided in Figure 2.1. The matrix $H$ is known as the *Hadamard* matrix and the matrices $X$, $Y$, and $Z$ are known as the *Pauli* matrices.

A complex matrix $U$ is *Hermitian* if $U = U^\dagger$. If $U$ is hermitian, then $u^\dagger U u$ is always real. Note that all the matrices in Figure 2.1 are Hermitian.

A matrix $U$ is *positive semidefinite* (resp. *positive definite*) if $U$ is hermitian and $\alpha^\dagger U\alpha \geqslant 0$ (resp. $\alpha^\dagger U\alpha > 0$) for all $\alpha \in \mathbb{C}^n$.

### 2.1.4 Tensor products

The *tensor product* of vectors spaces and matrices is defined as usual and denoted by $\otimes$. We note that $\mathbb{C}^n \otimes \mathbb{C}^m = \mathbb{C}^{nm}$ and that the tensor product acts on square matrices as

$$\otimes : \mathcal{M}_{n,n}(\mathbb{C}) \times \mathcal{M}_{m,m}(\mathbb{C}) \to \mathcal{M}_{nm,nm}(\mathbb{C}).$$

Given two vectors $\alpha \in \mathbb{C}^n$, $\beta \in \mathbb{C}^m$, their tensor product $\gamma = \alpha \otimes \beta \in \mathbb{C}^{nm}$ is defined by $\gamma_{(i,j)} = \alpha_i \beta_j$, with the pairs $(i, j)$ ordered lexicographically. One obtains a basis for the tensor product $\mathcal{H} \otimes \mathcal{H}'$ of two vector spaces $\mathcal{H}$ and $\mathcal{H}'$ by tensoring the elements of the bases for $\mathcal{H}$ and $\mathcal{H}'$. For example, if $\{\alpha_1, \alpha_2\}$ and $\{\beta_1, \beta_2\}$ are two bases for $\mathbb{C}^2$, then

$$\{\alpha_1 \otimes \beta_1, \alpha_1 \otimes \beta_2, \alpha_2 \otimes \beta_1, \alpha_2 \otimes \beta_2\}$$

forms a basis for $\mathbb{C}^2 \otimes \mathbb{C}^2 = \mathbb{C}^4$. We note, however, that not all elements of $\mathbb{C}^4$ are of the form $\alpha \otimes \beta$ with $\alpha, \beta \in \mathbb{C}^2$.

## 2.2 Quantum Computation

### 2.2.1 A single quantum bit

Recall that the fundamental unit of classical computation is the *bit*. By analogy, the fundamental unit of quantum computation is called the *quantum bit*, or *qubit*.

**Definition 2.2.1.** The *state of a qubit* is a unit vector in $\mathbb{C}^2$ considered *up to a phase*, that is, up to multiplication by a unit vector of $\mathbb{C}$.

As is customary, we use the so-called *ket* notation to denote the state of a qubit, which is written $|\phi\rangle$. Moreover, we write $|0\rangle$ and $|1\rangle$ for the elements $e_1$ and $e_2$ of the standard basis for $\mathbb{C}^2$. In the context of quantum computation, the basis $\{|0\rangle, |1\rangle\}$ is referred to as the *computational basis*. Since the state of a (classical) bit is an element of the set $\{0, 1\}$, we sometimes refer to the states $|0\rangle = 1|0\rangle + 0|1\rangle$ and $|1\rangle = 0|0\rangle + 1|1\rangle$ as the *classical states*.

By Definition 2.2.1, the state of a qubit can be one of the classical states but also any linear combination $\alpha|0\rangle + \beta|1\rangle$ such that $||\alpha||^2 + ||\beta||^2 = 1$. The coefficients $\alpha$ and $\beta$ in such a linear combination are called the *amplitudes* of the state. When

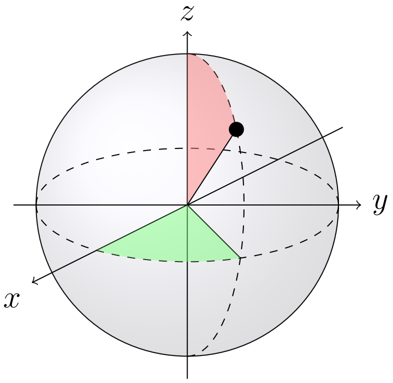


Figure 2.2: The Bloch sphere representation of the state of a qubit. The state $(\theta, \phi)$ is represented by the black dot, with $\theta$ corresponding to the angle pictured in red and $\phi$ to the angle pictured in green.

both amplitudes of the state of a qubit are non-zero, then the qubit is said to be in a *superposition* of $|0\rangle$ and $|1\rangle$.

The fact that the vector in Definition 2.2.1 is considered only up to a phase gives rise to a nice geometric representation of the state of a qubit. Let $\alpha|0\rangle + \beta|1\rangle$ be a unit vector. We can rewrite this linear combination as

$$\alpha|0\rangle + \beta|1\rangle = e^{i\gamma}(\cos(\frac{\theta}{2})|0\rangle + e^{i\phi}\sin(\frac{\theta}{2})|1\rangle)$$

for some real numbers $\theta \in [0, \pi]$ and $\gamma, \phi \in [0, 2\pi]$. Since the state of qubit is defined up to a phase, the same state is described by

$$\cos(\frac{\theta}{2})|0\rangle + e^{i\phi}\sin(\frac{\theta}{2})|1\rangle$$

with $\theta, \phi \in \mathbb{R}$. The pair $(\theta, \phi)$ defines a point on the 2-sphere known as the *Bloch sphere* representation of the state, as pictured in Figure 2.2. The Cartesian coordinates of the point $(\theta, \phi)$ on the Bloch sphere are given by $(\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$.

### 2.2.2 Multiple quantum bits

In classical computation, the state of a system of $n$ bits is represented by an element of the set $\{0, 1\}^n$. The set of states of a complex system therefore arises as a *Cartesian product*. In contrast, the set of states of a system composed of multiple qubits is obtained using the *tensor product*.

**Definition 2.2.2.** The *state of a system of $n$ qubits* is a unit vector in $\mathbb{C}^{2^n}$ considered up to a phase.

The basis $\{|0\rangle, |1\rangle\}$ for $\mathbb{C}^2$ can be used to construct a basis for $\mathbb{C}^{2^n}$, which we also call the computational basis. As is customary, we denote the basis element $|x_1\rangle \otimes \ldots \otimes |x_n\rangle$ by $|x_1 \ldots x_n\rangle$, for any $x_1, \ldots, x_n \in \{0, 1\}$. For example, the state of a system of two qubits is described, up to a phase, by a linear combination

$$\alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle$$

for some complex numbers $\alpha_i$ such that $\sum ||\alpha_i||^2 = 1$. As mentioned in Section 2.1.4, not all elements of $\mathbb{C}^4$ arise as the tensor of two elements of $\mathbb{C}^2$. If the state of a two-qubit system can be written as

$$(\alpha|0\rangle + \beta|1\rangle) \otimes (\alpha'|0\rangle + \beta'|1\rangle)$$

then the two qubits are said to be *separable.* Otherwise, they are said to be *entangled.*

### 2.2.3 Evolution of a quantum system

A computation is performed by acting on the state of a system of qubits. This can be done in two ways: via *unitary transformation* or via *measurement.*

#### Unitary evolution

The state of a system of $n$ qubits can evolve under the action of a unitary operator. If $|\phi\rangle$ is such a state (viewed as a column vector in $\mathbb{C}^{2^n}$) and $U$ is a unitary matrix, the evolution of $|\phi\rangle$ under $U$ is given by $|\phi\rangle \mapsto U|\phi\rangle$. For this reason, we sometimes refer to a unitary matrix of dimension $2^n$ as an *$n$-qubit unitary.*

Recall from Section 2.2.1 that the state of a qubit can be interpreted as a point on the Bloch sphere. This interpretation extends to single-qubit unitary matrices, which can be seen as *rotations* of the Bloch sphere. Let $v = (x, y, z)$ be a unit vector in $\mathbb{R}^3$ and $\theta \in \mathbb{R}$ and define the matrix

$$R_v(\theta) = \cos(\frac{\theta}{2})I - \sin(\frac{\theta}{2})(xX + yY + zZ),$$

where $X$, $Y$, and $Z$ are the Pauli matrices. The matrix $R_v(\theta)$ defines a rotation of the Bloch sphere by $\theta$ radians about the $v$-axis. For example, the Pauli matrices $X$,

$Y$, and $Z$ correspond to rotations about the $x$-, $y$-, and $z$-axes by $\pi$ radians. The following theorem states that, up to a phase, every unitary can be seen as a rotation of the Bloch sphere.

**Theorem 2.2.3.** *If $U \in U(2)$, then there exist real numbers $\alpha$ and $\theta$, and a unit vector $w$ in $\mathbb{R}^3$ such that $U = e^{i\alpha} R_w(\theta)$.*

### Measurement

The second way in which one can act on the state of a system of qubits is by measurement. Unlike unitary evolutions, measurements are probabilistic processes.

We first describe the effect of a measurement on a single qubit. Assume a qubit is in the state $\alpha|0\rangle + \beta|1\rangle$. If the qubit is measured, then the result of the measurement is either 0 or 1 and the state of the qubit post-measurement is the corresponding classical state. Moreover, the measurement result 0 occurs with probability $||\alpha||^2$ and the measurement result 1 occurs with probability $||\beta||^2$. Since the state of a qubit was described by a unit vector, these probabilities sum to 1.

We now discuss the case of a complex system. For simplicity, we only consider a two-qubit system. Assume the state of our system is given by the following vector

$$\alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle.$$

If the first qubit is measured, then with probability $||\alpha_0||^2 + ||\alpha_1||^2$ the measurement result is 0 and the post-measurement state is

$$\frac{\alpha_0}{\sqrt{||\alpha_0||^2 + ||\alpha_1||^2}}|00\rangle + \frac{\alpha_1}{\sqrt{||\alpha_0||^2 + ||\alpha_1||^2}}|01\rangle,$$

while with probability $||\alpha_2||^2 + ||\alpha_3||^2$ the measurement result is 1 and the post-measurement state is

$$\frac{\alpha_2}{\sqrt{||\alpha_2||^2 + ||\alpha_3||^2}}|10\rangle + \frac{\alpha_3}{\sqrt{||\alpha_2||^2 + ||\alpha_3||^2}}|11\rangle.$$

Note that the linear combinations have been renormalized to ensure that the resulting states are described by unit vectors.

### No-cloning

The well-known *no-cloning theorem* is a property of quantum computation which will be of importance in chapters 4, 8, and 9. The theorem states that there is no physical device whose action is described by the following mapping

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

In other words, it is impossible to “clone” quantum states.

## 2.2.4 The QRAM model and quantum circuits

In Section 2.2, we described the mathematical formalism of quantum computation, but did not spend any time explaining what a quantum computer might look like, nor how one would describe quantum programs and protocols. Various models of quantum computation have been devised in the literature (see, e.g., [15], [52]). Here, we discuss two complementary approaches: the *QRAM model* introduced by Knill in [44] and the *circuit model* described by Deutsch in [16]. We also refer the reader to [1] where this model was described in more detail.

### The QRAM model of quantum computation

In the QRAM model of a quantum computation, a quantum computer is thought of as consisting of two devices, a *classical device* and a *quantum device*, sharing computational tasks. The classical device performs operations such as compilation and correctness checking. The quantum device only performs the specifically quantum operations. In particular, it is assumed to hold an array of qubits and to be able to

- initialize qubits to a specified state,
- perform unitary operations on qubits, and
- measure qubits.

The execution of a program in this model proceeds as follows. The source code of the program resides on the classical device where it is compiled into an executable. When the program is executed on the classical device, it can communicate with the quantum device if required. Through this communication, it can instruct the quantum device

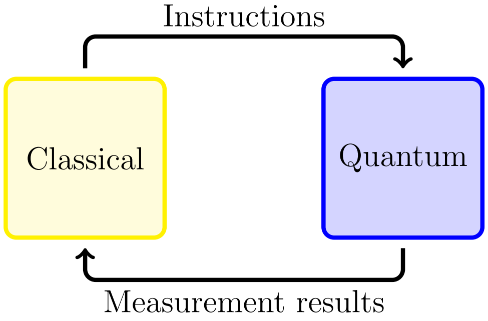


Figure 2.3: The QRAM model of quantum computation.

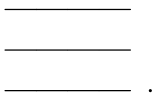
to perform one of the above described quantum operations. Measurement results, if any, are returned to the classical device for post-processing or further computation. This system is schematically represented in Figure 2.3.

### Quantum circuits

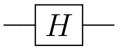
While the QRAM model of quantum computation describes the overall architecture of a quantum computer, quantum circuits are a language to express (parts of) quantum algorithms. In particular, one can think of a quantum circuit as the description of a sequence of operations that, in the QRAM model, the classical device would send to its quantum counterpart.

In the quantum circuit model, a quantum computation is thought of as a sequence of unitary gates applied to an array of qubits, followed by a measurement of some or all of the qubits. The sequence of unitary operations are arranged in the form of a circuit, akin to the classical boolean circuits.

The identity matrix on $n$ qubits, i.e., the identity matrix of dimension $2^n$, is represented by $n$ distinct *wires*. For example, the identity on 3 qubits is represented by



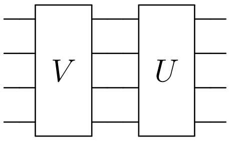
A non-identity unitary $U$ acting on $n$ qubits is depicted as a box, labelled $U$, with $n$ *input wires* and $n$ *output wires*. For example, the Hadamard matrix $H$ is represented as



.

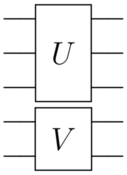
Because of the similarities between quantum circuits and classical boolean circuits, we sometimes refer to a unitary $U$ as a *quantum gate*.

The graphical representation of the composition $UV$, of two unitary matrices $U$ and $V$ acting on $n$ qubits, is obtained by horizontally concatenating the gates for $U$ and $V$. That is, by connecting the output wires of the gate for $V$ to the input wires of the gate for $U$, as illustrated below in the case of matrices on 4 qubits.



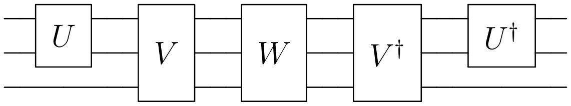
.

Finally, the graphical representation of the tensor product $U \otimes V$ of two unitary matrices $U$ and $V$ is obtained by vertically concatenating the gates for $U$ and $V$, as illustrated below in the case of unitary matrices $U$ and $V$ on 3 and 2 qubits respectively.



.

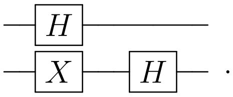
Now let $S$ be a set of unitary matrices and write $S^\dagger$ for the set $\{U^\dagger\ ;\ U \in S\}$. A *circuit over* $S$ is constructed using the gates of $S \cup S^\dagger$, as well as arbitrary identity gates, using the graphical representations for composition and tensor product. For example, a circuit over $S = \{U, V, W\}$, where $U$ acts on 2 qubits and $V$ and $W$ both act on 3 qubits, is represented below



.

We write $\langle S \rangle$ for the set of all circuits over $S$.

One can recover the matrix represented by a circuit by interpreting the operations of horizontal and vertical concatenation as composition and tensor product respectively. For example, let $S$ consist of the Hadamard gate $H$ and the Pauli gate $X$, and let $C$ be the following circuit



Then $C$ represents the unitary matrix $U$ given by

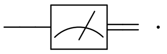
$$U = (I_2 \otimes H) \circ (H \otimes X) = \frac{1}{2} \begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \end{bmatrix}.$$

Note that $C$ could have alternatively been interpreted as the unitary $(I_2 \circ H) \otimes (H \circ X)$. However, both interpretations coincide since $(I_2 \circ H) \otimes (H \circ X) = (I_2 \otimes H) \circ (H \otimes X)$. This is due to the so-called *bifunctoriality* of $\otimes$, which guarantees in particular that

$$(U \circ U') \otimes (V \circ V') = (U \otimes V) \circ (U' \otimes V')$$

for any $U, U', V, V'$. By a slight abuse of notation, we often write $C = U$ if the circuit $C$ represents the matrix $U$.

The language of quantum circuits can be extended with measurement gates, which are depicted as



# Chapter 3

# Algebraic number theory

In this chapter, we introduce basic concepts of algebraic number theory. In particular, we describe ring extensions of $\mathbb{Z}$ and computational methods to solve certain Diophantine equations, known as *relative norm equations*, over these rings. References for this material include [49] and [13].

## 3.1 Rings of integers

If $K$ is a field, $S$ a subfield of $K$, and $\alpha$ an element of $K$, then the *field extension* $S(\alpha)$ is the smallest subfield of $K$ which contains $S$ and $\alpha$.

An element $\alpha \in \mathbb{C}$ is an *algebraic number* if it is the root of some polynomial over $\mathbb{Q}$. A field extension of $\mathbb{Q}$ of the form $\mathbb{Q}(\alpha)$ for some algebraic number $\alpha$ is called an *algebraic number field*.

An element $\beta \in \mathbb{C}$ is an *algebraic integer* if it is the root of some monic polynomial over $\mathbb{Z}$, i.e., of some polynomial over $\mathbb{Z}$ whose leading coefficient is 1. The set of algebraic integers of a number field $\mathbb{Q}(\alpha)$ forms a ring, called *the ring of integers* of $\mathbb{Q}(\alpha)$ and denoted by $\mathcal{O}_{\mathbb{Q}(\alpha)}$. For any algebraic number field $\mathbb{Q}(\alpha)$, the ring $\mathcal{O}_{\mathbb{Q}(\alpha)}$ is an integral domain whose field of fractions is $\mathbb{Q}(\alpha)$.

### 3.1.1 Extensions of $\mathbb{Z}$

If $R$ is a ring, $R'$ a subring of $R$, and $\alpha$ an element of $R$, then the *ring extension* $R'[\alpha]$ is the smallest subring of $R$ which contains $R'$ and $\alpha$.

**Definition 3.1.1** (Extensions of $\mathbb{Z}$)**.** We are interested in the following four ring extensions of $\mathbb{Z}$.

- The ring $\mathbb{Z}$ of *integers*.

- The ring $\mathbb{Z}[i]$ of *Gaussian integers*.

- The ring $\mathbb{Z}[\sqrt{2}]$ of *quadratic integers with radicand 2*.
- The ring $\mathbb{Z}[\omega]$ of *cyclotomic integers of degree 8*, where $\omega = e^{i\pi/4} = (1+i)/\sqrt{2}$.

We note that since $i = \omega^2$ and $\sqrt{2} = \omega - \omega^3$, we have the inclusions $\mathbb{Z} \subseteq \mathbb{Z}[i] \subseteq \mathbb{Z}[\omega]$ and $\mathbb{Z} \subseteq \mathbb{Z}[\sqrt{2}] \subseteq \mathbb{Z}[\omega]$. Moreover, one can show that $\mathbb{Z}[\sqrt{2}]$ and $\mathbb{Z}[\omega]$ are dense in $\mathbb{R}$ and $\mathbb{C}$ respectively. The rings introduced in Definition 3.1.1 are rings of algebraic integers. Indeed we have

$$\mathbb{Z} = \mathcal{O}_{\mathbb{Q}},\ \mathbb{Z}[i] = \mathcal{O}_{\mathbb{Q}(i)},\ \mathbb{Z}[\sqrt{2}] = \mathcal{O}_{\mathbb{Q}(\sqrt{2})},\text{ and } \mathbb{Z}[\omega] = \mathcal{O}_{\mathbb{Q}(\omega)}.$$

Explicit expressions for the elements of $\mathbb{Z}[i]$, $\mathbb{Z}[\sqrt{2}]$, and $\mathbb{Z}[\omega]$ are given below.

- $\mathbb{Z}[i] = \{a_0 + a_1 i \ ;\ a_j \in \mathbb{Z}\}$.
- $\mathbb{Z}[\sqrt{2}] = \{a_0 + a_1\sqrt{2} \ ;\ a_j \in \mathbb{Z}\}$.
- $\mathbb{Z}[\omega] = \{a_0 + a_1\omega + a_2\omega^2 + a_3\omega^3 \ ;\ a_j \in \mathbb{Z}\}$.

Note that there is a bijection between $\mathbb{Z}[i]$ and $\mathbb{Z}^2$. As the following proposition shows, there is also a bijection between $\mathbb{Z}[\omega]$ and and two disjoint copies of $\mathbb{Z}[\sqrt{2}] \times \mathbb{Z}[\sqrt{2}]$.

**Proposition 3.1.2.** *A complex number $\alpha$ is in $\mathbb{Z}[\omega]$ if and only if it can be written of the form $\alpha = a_0 + a_1 i$ or of the form $\alpha = a_0 + a_1 i + \omega$, where $a_0, a_1 \in \mathbb{Z}[\sqrt{2}]$.*

*Proof.* The right-to-left implication is trivial. For the left-to-right implication, let $\alpha = a + b\omega + c\omega^2 + d\omega^3$, where $a, b, c, d \in \mathbb{Z}$. Noting that $\omega = \frac{1+i}{\sqrt{2}}$, we have

$$\alpha = (a + \frac{b-d}{2}\sqrt{2}) + (c + \frac{b+d}{2}\sqrt{2})i.$$

If $b-d$ (and therefore $b+d$) is even, then $\alpha$ is of the first form, with $a_0 = a + \frac{b-d}{2}\sqrt{2}$ and $a_1 = c + \frac{b+d}{2}\sqrt{2}$. If $b-d$ (and therefore $b+d$) is odd, then $\alpha$ is of the second form, with $a_0 = a + \frac{b-d-1}{2}\sqrt{2}$ and $a_1 = c + \frac{b+d-1}{2}\sqrt{2}$. □

We close this subsection with the definition of two algebraic integers which will be useful in chapters 5, 6, and 7.

**Definition 3.1.3.** The algebraic integers $\lambda \in \mathbb{Z}[\sqrt{2}]$ and $\delta \in \mathbb{Z}[\omega]$ are defined as follows

- $\lambda = 1 + \sqrt{2}$ and
- $\delta = 1 + \omega$.

### 3.1.2 Automorphisms

Recall that an automorphism of a ring $R$ is an isomorphism $R \to R$. The ring $\mathbb{Z}[\omega]$ has four automorphisms. One of these automorphisms is *complex conjugation*, which we denote $(-)^\dagger$ as in Chapter 2. Explicitly, $(-)^\dagger$ acts on an arbitrary element of $\mathbb{Z}[\omega]$ as follows

$$(a_0 + a_1\omega + a_2\omega^2 + a_3\omega^3)^\dagger = a_0 - a_3\omega - a_2\omega^2 - a_1\omega^3.$$

A second automorphism of $\mathbb{Z}[\omega]$ is $\sqrt{2}$-*conjugation*, denoted $(-)^\bullet$, which acts on an arbitrary element of $\mathbb{Z}[\omega]$ as follows

$$(a_0 + a_1\omega + a_2\omega^2 + a_3\omega^3)^\bullet = a_0 - a_1\omega + a_2\omega^2 - a_3\omega^3.$$

The remaining two automorphisms of $\mathbb{Z}[\omega]$ are the identity as well as $(-)^{\bullet\dagger} = (-)^{\dagger\bullet}$.

The rings $\mathbb{Z}[i]$ and $\mathbb{Z}[\sqrt{2}]$ both have two automorphisms, while $\mathbb{Z}$ has exactly one. All of these are obtained by restricting the automorphisms of $\mathbb{Z}[\omega]$. Because $(-)^\dagger$ acts trivially on $\mathbb{Z}[\sqrt{2}]$, the only non-identity automorphism of $\mathbb{Z}[\sqrt{2}]$ is $(-)^\bullet$. Explicitly, the action of $(-)^\bullet$ on an element of $\mathbb{Z}[\sqrt{2}]$ is given by $(a+b\sqrt{2})^\bullet = a-b\sqrt{2}$. Similarly, the only non-identity automorphism of $\mathbb{Z}[i]$ is $(-)^\dagger$, whose action is explicitly given by $(a+bi)^\dagger = a - bi$. The ring $\mathbb{Z}$ has no non-trivial automorphism.

We note that for $t \in \mathbb{Z}[\omega]$, we have $t \in \mathbb{Z}[\sqrt{2}]$ iff $t = t^\dagger$, $t \in \mathbb{Z}[i]$ iff $t = t^\bullet$, and $t \in \mathbb{Z}$ iff $t = t^\dagger$ and $t = t^\bullet$.

### 3.1.3 Norms

Let $R$ be one of the rings $\mathbb{Z}$, $\mathbb{Z}[i]$, $\mathbb{Z}[\sqrt{2}]$, or $\mathbb{Z}[\omega]$. We define the *norm* $\mathcal{N}_R(\alpha)$ of an element $\alpha \in R$ to be

$$\mathcal{N}_R(\alpha) = \prod_\sigma \sigma(\alpha),$$

where the product is taken over all automorphisms $\sigma : R \to R$. We provide explicit formulas for each norm in the definition below.

**Definition 3.1.4** (Norms).

- If $\alpha \in \mathbb{Z}$, then $\mathcal{N}_\mathbb{Z}(\alpha) = \alpha$.

- If $\alpha = a_0 + a_1 i \in \mathbb{Z}[i]$, then $\mathcal{N}_{\mathbb{Z}[i]}(\alpha) = \alpha^\dagger\alpha = a_0^2 + a_1^2$.

- If $\alpha = a_0 + a_1\sqrt{2} \in \mathbb{Z}[\sqrt{2}]$, then $\mathcal{N}_{\mathbb{Z}[\sqrt{2}]}(\alpha) = \alpha^\bullet\alpha = a_0^2 - 2a_1^2$.
- If $\alpha = a_0 + a_1\omega + a_2\omega^2 + a_3\omega^3 \in \mathbb{Z}[\omega]$, then

$$\mathcal{N}_{\mathbb{Z}[\omega]}(\alpha) = \alpha^{\dagger\bullet}\alpha^\dagger\alpha^\bullet\alpha = (a_0^2 + a_1^2 + a_2^2 + a_3^2)^2 - 2(a_3a_2 + a_2a_1 + a_1a_0 - a_3a_0)^2.$$

All the norms introduced in Definition 3.1.4 are multiplicative and integer valued. This means that $\mathcal{N}_R(\alpha\beta) = \mathcal{N}_R(\alpha)\mathcal{N}_R(\beta)$ and $\mathcal{N}_R(\alpha) \in \mathbb{Z}$. The norms $\mathcal{N}_{\mathbb{Z}[\omega]}$ and $\mathcal{N}_{\mathbb{Z}[i]}$ are moreover valued in the non-negative integers. Finally, we have $\mathcal{N}_R(\alpha) = 0$ iff $\alpha = 0$ and $\mathcal{N}_R(\alpha)$ is a unit if and only if $\alpha$ is a unit, that is, an invertible element.

*Remark* 3.1.5. If $\alpha$ and $\beta$ are two distinct elements of $\mathbb{Z}[\sqrt{2}]$, then the following inequality holds:

$$|\alpha - \beta| \cdot |\alpha^\bullet - \beta^\bullet| \geqslant 1, \tag{3.1}$$

This follows from the fact that $|\alpha-\beta|\cdot|\alpha^\bullet-\beta^\bullet| = |\mathcal{N}_{\mathbb{Z}[\sqrt{2}]}(\alpha-\beta)|$. The same inequality holds for $\alpha, \beta \in \mathbb{Z}[\omega]$.

## 3.2 Diophantine equations

### 3.2.1 Euclidean domains

The rings $\mathbb{Z}$, $\mathbb{Z}[i]$, $\mathbb{Z}[\sqrt{2}]$, and $\mathbb{Z}[\omega]$ are integral domains, whose fields of fractions are $\mathbb{Q}$, $\mathbb{Q}(i)$, $\mathbb{Q}(\sqrt{2})$, and $\mathbb{Q}(\omega)$. An important property of these rings is that they are *Euclidean domains.*

**Definition 3.2.1.** A *Euclidean domain* is an integral domain $R$ equipped with a function $f : R \setminus \{0\} \to \mathbb{N}$ such that for every $a \in R$ and $b \in R \setminus \{0\}$, there exist $q, r \in R$ such that $a = bq + r$ and $r = 0$ or $f(r) < f(b)$.

**Proposition 3.2.2.** *Let $R$ be one of $\mathbb{Z}$, $\mathbb{Z}[i]$, $\mathbb{Z}[\sqrt{2}]$, or $\mathbb{Z}[\omega]$. Then the function $|\mathcal{N}_R(-)|$ makes $R$ into a Euclidean domain.*

The notion of divisibility, as well as many essential properties of the divisibility of integers can be defined in an arbitrary Euclidean domain. In particular, we write $x \mid y$ if $x$ is a divisor of $y$, and $x \sim y$ if $x \mid y$ and $y \mid x$. An element $x$ is *prime* if $x$ is not a unit and $x = ab$ implies that either $a$ or $b$ is a unit. The notion of greatest common divisor, as well as Euclid's algorithm, can be defined in any Euclidean domain. Finally, every

Euclidean domain is also a *unique factorization domain*, which means that every non-zero non-unit element of the ring can be factored into primes in an essentially unique way.

### 3.2.2 Relative norm equations

In chapters 6 and 7, we will be interested in solving certain equations known as *relative norm equations*. Specifically, we will be concerned with the following two problems.

**Problem 3.2.3** (Relative norm equation over $\mathbb{Z}[i]$)**.** Given $\beta \in \mathbb{Z}$, find $\alpha \in \mathbb{Z}[i]$ such that $\alpha^\dagger \alpha = \beta$.

**Problem 3.2.4** (Relative norm equation over $\mathbb{Z}[\omega]$)**.** Given $\beta \in \mathbb{Z}[\sqrt{2}]$, find $\alpha \in \mathbb{Z}[\omega]$ such that $\alpha^\dagger \alpha = \beta$.

Solving Problem 3.2.3 amounts to finding $\alpha \in \mathbb{Z}[i]$ such that $\mathcal{N}_{\mathbb{Z}[i]}(\alpha) = \beta$. The equation to be solved is therefore a norm equation. In Problem 3.2.4, however, $\alpha^\dagger \alpha \neq \mathcal{N}_{\mathbb{Z}[\omega]}(\alpha)$. In this case, we do not consider all the automorphic images of $\alpha$. Instead, we consider the automorphic images of $\alpha$ under the automorphisms of $\mathbb{Z}[\omega]$ that fix $\mathbb{Z}[\sqrt{2}]$. For this reason the equation in Problem 3.2.4 is a *relative* norm equation. For uniformity, we refer to both equations as relative norm equations.

A *Diophantine equation* is a polynomial equation in integer variables. By writing $\alpha = a + bi$ with $a, b \in \mathbb{Z}$, Problem 3.2.3 becomes equivalent to the Diophantine equation

$$a^2 + b^2 = \beta.$$

Similarly, by writing $\alpha = a + b\omega + c\omega^2 + d\omega^3$ and $\beta = a' + b'\sqrt{2}$ with $a, a', b, b', c, d \in \mathbb{Z}$, Problem 3.2.4 becomes equivalent to the system of Diophantine equations

$$\begin{aligned} a^2 + b^2 + c^2 + d^2 &= a' \\ ab - ad + cb + cd &= b'. \end{aligned}$$

In light of these equivalences, we sometimes abuse terminology and refer to the equations of problems 3.2.3 and 3.2.4 as Diophantine equations.

*Remark* 3.2.5. Problems 3.2.3 and 3.2.4 are computational problems. This means that a solution to either of these problems is an algorithm which decides whether the given equation has a solution and produces a solution if one exists. The algorithms solving problems 3.2.3 and 3.2.4 that we consider here are *probabilistic* in the sense that they make certain choices at random. We evaluate the time-complexity of an algorithm by estimating the number of *arithmetic operations* it requires to solve one the above problems. By arithmetic operations, we mean addition, subtraction, multiplication, division, exponentiation, and logarithm. When we say that an algorithm runs in probabilistic polynomial time, we mean that the algorithm is probabilistic, requires a polynomial number of expected arithmetic operations to solve the given problem, and produces a correct solution with probability greater than 1/2.

Since $\alpha^\dagger\alpha \geqslant 0$, a necessary condition for Problem 3.2.3 to have a solution is $\beta \geqslant 0$. Similarly, necessary conditions for Problem 3.2.4 to have a solution are $\beta \geqslant 0$ and $\beta^\bullet \geqslant 0$. This follows from the fact that $\alpha^\dagger\alpha = \beta$ implies $(\alpha^\bullet)^\dagger(\alpha^\bullet) = \beta^\bullet$. There are also sufficient conditions for the above relative norm equations to have solutions.

**Proposition 3.2.6.** *Let $\beta \in \mathbb{Z}$ be such that $\beta \geqslant 0$. If $\beta$ is prime and $\beta \equiv 1 \pmod 4$, then the equation $\alpha^\dagger\alpha = \beta$ has a solution.*

**Proposition 3.2.7.** *Let $\beta \in \mathbb{Z}[\sqrt{2}]$ be such that $\beta \geqslant 0$ and $\beta^\bullet \geqslant 0$ and let $n = \beta^\bullet\beta \in \mathbb{Z}$. If $n$ is prime and $n \equiv 1 \pmod 8$, then the equation $\alpha^\dagger\alpha = \beta$ has a solution.*

We close this chapter by stating that there are solutions to problems 3.2.3 and 3.2.4.

**Proposition 3.2.8.** *Let $\beta \in \mathbb{Z}$. Given the prime factorization of $\beta$, there exists an algorithm that determines, in probabilistic polynomial time, whether the equation $\alpha^\dagger\alpha = \beta$ has a solution $\alpha \in \mathbb{Z}[i]$ or not, and finds a solution if there is one.*

**Proposition 3.2.9.** *Let $\beta \in \mathbb{Z}[\sqrt{2}]$, and let $n = \beta^\bullet\beta$. Given the prime factorization of $n$, there exists an algorithm that determines, in probabilistic polynomial time, whether the equation $\alpha^\dagger\alpha = \beta$ has a solution $\alpha \in \mathbb{Z}[\omega]$ or not, and finds a solution if there is one.*

# Chapter 4

# The lambda calculus

In this chapter, we introduce the untyped lambda calculus as well as various typed lambda calculi, including the quantum lambda calculus. The standard reference for the untyped lambda calculus is [4]. For typed lambda calculi, see [24]. For the quantum lambda calculus, see [64], [61], or [58].

## 4.1 The untyped lambda calculus

### 4.1.1 Concrete terms

We start by defining the *syntax* of the untyped lambda calculus.

**Definition 4.1.1.** The *concrete terms* of the untyped lambda calculus are defined by

$$a, b \quad ::= \quad x \;\Big|\; (\lambda x.a) \;\Big|\; (ab)$$

where $x$ comes from a countable set $\mathcal{V}$ of *variables.*

In Definition 4.1.1, concrete terms are given in the so-called *Backus-Naur Form.* This notation should be interpreted as defining the collection $L$ of all concrete terms as the smallest set of words on the alphabet $\mathcal{V} \cup \{(, ), \lambda, .\}$ such that

- $\mathcal{V} \subseteq L$,
- if $a \in L$ and $x \in \mathcal{V}$, then $(\lambda x.a) \in L$, and
- if $a, b \in L$, then $(ab) \in L$.

A concrete term of the form $(\lambda x.a)$ is called a *lambda abstraction* and we say that $a$ is the *body* of the abstraction. A concrete term of the form $(ab)$ is called an *application.* The operations of lambda abstraction and application are called *term forming operations.*

To increase the readability of concrete terms, we adopt the following notational conventions

- outermost parentheses are omitted,
- applications associate to the left,
- the body of a lambda abstraction extends as far to the right as possible, and
- multiple lambda abstractions are contracted.

This implies, for example, that the term $(\lambda x.(\lambda y.((xy)x)))$ will be written $\lambda xy.xyx$.

The intended interpretation of concrete terms is as follows. The lambda abstraction $\lambda x.a$ represents the function defined by the rule $x \mapsto a$. The application $ab$ represents the application of the function $a$ to the argument $b$.

As a first example, consider the identity function. Since it acts as $x \mapsto x$, its representation as a concrete term should be $\lambda x.x$. The application of the identity function to some input $a$ is written as $(\lambda x.x)a$. As a second example, consider the function that acts as $x \mapsto (y \mapsto x)$. This function inputs $x$ and outputs the constant function to $x$. Its representation as a concrete term is $\lambda xy.x$.

### 4.1.2 Reduction

We now define the *operational semantics* of the terms of the untyped lambda calculus. That is, we attribute meaning to the terms by specifying their behavior.

For the concrete term $\lambda x.x$ to be an acceptable representative for the identity function, it should "behave" accordingly, i.e., the concrete term $(\lambda x.x)a$ should reduce to $a$. One way to achieve this is to define the reduction relation by

$$(\lambda x.b)a \to b[a/x], \tag{4.1}$$

where $b[a/x]$ stands for the substitution of $a$ for every occurrence of $x$ in $b$. Under this definition of the reduction relation we have $(\lambda x.x)a \to a$, as intended.

Now consider the concrete term $\lambda xy.x$. Under the interpretation of terms given above, it represents the function $x \mapsto (y \mapsto x)$. If we apply this concrete term to $x'y'$, then (4.1) yields the expected result

$$(\lambda xy.x)(x'y') \to (\lambda y.x)[(x'y')/x] = \lambda y.x'y'.$$

However, if we apply $\lambda xy.x$ to the variable $y$ and reduce according to (4.1) again, we get

$$(\lambda xy.x)y \to (\lambda y.x)[y/x] = \lambda y.y.$$

This is unsatisfactory because the concrete term $\lambda y.y$ represents the identity function, not the constant function to $y$. The way around this problem is to start by renaming $\lambda xy.x$ to, say, $\lambda xz.x$. Under this renaming, the reduction of (4.1) would produce the concrete term $\lambda z.y$, representing a constant function to $y$. We therefore need to define a substitution method that appropriately renames variables.

**Definition 4.1.2.** Let $x$ and $y$ be two variables and $a$ be a concrete term. The *renaming of $x$ by $y$ in $a$*, written $a\{y/x\}$, is defined as

- $x\{y/x\} = y$,
- $z\{y/x\} = z$ if $x \neq z$,
- $ab\{y/x\} = (a\{y/x\})(b\{y/x\})$,
- $\lambda x.a\{y/x\} = \lambda y.(a\{y/x\})$, and
- $\lambda z.a\{y/x\} = \lambda z.(a\{y/x\})$ if $x \neq z$.

**Definition 4.1.3.** The set of *free variables* of a concrete term $a$, written $\mathrm{FV}(a)$, is defined as

- $\mathrm{FV}(x) = \{x\}$,
- $\mathrm{FV}(ab) = \mathrm{FV}(a) \cup \mathrm{FV}(b)$, and
- $\mathrm{FV}(\lambda x.a) = \mathrm{FV}(a) \setminus \{x\}$.

Any variable that appears in $a$ but does not belong to $\mathrm{FV}(a)$ is said to be *bound*. For this reason, we call $\lambda$ a *binder*.

**Definition 4.1.4.** If $a$ is a concrete term such that $\mathrm{FV}(a) = \emptyset$, then $a$ is said to be *closed*.

In the terminology of Definition 4.1.3, the problem with the relation defined by (4.1) is that the variable $y$ is free on the left of $\to$ but bound on the right. The variable $y$ is said to have been *captured* in the course of the reduction. Hence, we introduce a *capture-avoiding* notion of substitution.

**Definition 4.1.5.** Let $x$ be a variable, and $a$ and $b$ be two concrete terms. The *substitution of* $b$ *for* $x$ *in* $a$, written $a[b/x]$, is defined as

- if $a = x$, then $a[b/x] = b$,
- if $a = y$ and $y \neq x$, then $a[b/x] = a$,
- if $a = cc'$, then $a[b/x] = c[b/x]c'[b/x]$,
- if $a = \lambda x.c$, then $a[b/x] = a$, and
- if $a = \lambda y.c$, then $a[b/x] = \lambda z.(c\{z/y\})[b/x]$ where $z \notin \mathrm{FV}(b) \cup \mathrm{FV}(c)$.

*Remark* 4.1.6. The last clause of Definition 4.1.5 is slightly ambiguous. Indeed, the variable $z$ is not specified, but only required to belong to $\mathcal{V} \setminus (\mathrm{FV}(b) \cup \mathrm{FV}(c))$. We say of such a $z$ that it is *fresh with respect to* $b$ *and* $c$. To be more rigorous, we should require the set $\mathcal{V}$ to be well-ordered and choose $z$ to be the least element of $\mathcal{V} \setminus (\mathrm{FV}(b) \cup \mathrm{FV}(c))$. Since this ambiguity will be lifted below when we move from concrete terms to *abstract* ones, we leave the definition unchanged.

To formally define the reduction relation, we start by identifying the strings, within a concrete term, that will give rise to a reduction. As one might expect, a reduction will take place whenever a function is applied to an argument.

**Definition 4.1.7.** A *redex* is a concrete term of the form $(\lambda x.a)b$. By extension, a *redex of a concrete term* $c$ is a redex that appears in $c$.

**Definition 4.1.8.** The *one-step* $\beta$*-reduction*, written $\to$, is defined on concrete terms by the rules

$$\frac{}{(\lambda x.a)b \to a[b/x]} \qquad \frac{a \to a'}{ab \to a'b} \qquad \frac{b \to b'}{ab \to ab'} \qquad \frac{a \to a'}{\lambda x.a \to \lambda x.a'} \,.$$

The $\beta$*-reduction*, written $\twoheadrightarrow$, is the reflexive and transitive closure of $\to$.

*Remark* 4.1.9. In Definition 4.1.8, $\twoheadrightarrow$ is defined as the *reflexive and transitive closure* of $\to$. This defines $\twoheadrightarrow$ as the smallest relation containing $\to$ that is reflexive and transitive.

With the reduction of concrete terms now defined, we can confirm that the concrete terms $\lambda x.x$ and $\lambda xy.x$ behave as expected since for any concrete term $a$ we have

$$(\lambda x.x)a \to a \quad \text{and} \quad (\lambda xy.x)a \to \lambda z.a$$

where $z \notin \mathrm{FV}(a) \cup \{x\}$.

### 4.1.3 Abstract terms

As noted in Remark 4.1.6, the reduction $(\lambda xy.x)a \to \lambda z.a$ depends on the choice of an ordering of the set $\mathcal{V}$, since $z$ is the least element of $\mathcal{V} \setminus (\mathrm{FV}(a) \cup \{x\})$. Thus, two different orderings of $\mathcal{V}$ will yield two different concrete terms $\lambda z.a$ and $\lambda z'.a$. This difference, however, is inessential since both terms define the same function $L \to L$ because $z, z' \notin \mathrm{FV}(a)$. The same inessential difference occurs between the concrete terms $\lambda z.z$ and $\lambda z'.z'$. To remove this distinction, we define an equivalence relation $=_\alpha$ on $L$ that equates the concrete terms differing only in the name of their bound variables. The idea behind this equivalence is that in the concrete term $\lambda x.a$, the occurrences of the variable $x$ in $a$ are place holders, rather than variables possessing an intrinsic identity. They can therefore be renamed without affecting the overall meaning of the concrete term.

**Definition 4.1.10.** The *$\alpha$-equivalence*, written $=_\alpha$, is defined on concrete terms as the smallest equivalence relation satisfying the rules

$$\frac{y \notin a}{\lambda x.a =_\alpha \lambda y.(a\{y/x\})} \qquad \frac{a =_\alpha a'}{ab =_\alpha a'b} \qquad \frac{b =_\alpha b'}{ab =_\alpha ab'} \qquad \frac{a =_\alpha a'}{\lambda x.a =_\alpha \lambda x.a'} \ .$$

where $y \notin a$ means that $y$ does not appear in $a$.

**Definition 4.1.11.** The *abstract terms* of the untyped lambda calculus are the elements of $L/{=_\alpha}$. We write $\Lambda$ for the set of all abstract terms.

If two concrete terms $a$ and $b$ are $\alpha$-equivalent, then they have the same structure (i.e., $a$ is an application if and only if $b$ is an application, and so on). For this reason, we keep writing $x$, $ab$, and $\lambda x.a$ for abstract terms even though we are dealing with equivalence classes of concrete terms.

When defining a function or a relation on abstract terms using the underlying concrete terms, we should make sure that this function or relation is well-defined. For example, if we define a function $F$ on abstract terms in this way, then we should verify that $a =_\alpha b$ implies $F(a) = F(b)$. One can check that the notions of free variable, capture-avoiding substitution and $\beta$-reduction are well-defined on abstract terms. In these cases, and in what follows, we generally overlook this obligation and omit the proofs of well-definedness.

Because from now on we will always be manipulating abstract terms, we refer to these as *terms* for brevity.

### 4.1.4 Properties of the untyped lambda calculus

An important property of the untyped lambda calculus is that it forms a complete model of computation in the sense of the Church-Turing thesis.

**Proposition 4.1.12.** *The untyped lambda calculus is Turing-complete, i.e., every Turing machine can be simulated by a lambda term.*

A term $a$ may have any number of redexes. If $a$ has no redexes, then the computation of $a$ is finished.

**Definition 4.1.13.** A term that does not contain any redexes is *reduced* or *in normal form.* If $b$ is reduced and $a \twoheadrightarrow b$ we say that $b$ is a normal form for $a$.

For example, a variable $x$ is reduced. Similarly, the terms $\lambda x.x$ and $\lambda xy.x$ are both reduced. The following term, on the other hand, is not reduced, since it contains two redexes

$$(\lambda x.x)((\lambda y.z)x').$$

When reducing such a term, nothing in the definition of the $\beta$-reduction tells us which redex to reduce first. We say that the $\beta$-reduction is *non-deterministic.*

**Proposition 4.1.14.** *The untyped lambda calculus is confluent, i.e., if $a$, $b$, and $b'$ are terms such that $a \twoheadrightarrow b$ and $a \twoheadrightarrow b'$, then there exists a term $c$ such that $b \twoheadrightarrow c$ and $b' \twoheadrightarrow c$.*

Proposition 4.1.14 was first established by Church and Rosser in [11] and is therefore known as the *Church-Rosser property.* It is also referred to as the *confluence* property of the $\beta$-reduction. Confluence guarantees that, despite the non-determinism of the reduction, normal forms are unique.

**Corollary 4.1.15.** *Let $a$ be a term. If $a$ has a normal form, then it is unique.*

The uniqueness of normal forms implies that the lambda calculus is *consistent* in the sense that not all terms are equated by the $\beta$-reduction.

**Corollary 4.1.16.** *The untyped lambda calculus is consistent, i.e., there exists two terms $a$ and $b$ such that $a \not\equiv_\beta b$, where $\equiv_\beta$ denotes the reflexive, symmetric, and transitive closure of $\to$.*

## 4.2 The simply typed lambda calculus

The term $xx$ is a well-formed term of the untyped lambda calculus. If we think of this term in light of the interpretation discussed in Section 4.1 we are led to interpret the variable $x$ as both function and argument in $xx$. This unusual construction is admitted in the untyped lambda calculus because there are no notions of domain and codomain for terms. *Types* can be seen as a method to endow the lambda calculus with these notions.

### 4.2.1 Terms

The language of the simply typed lambda calculus is an extension of the language of the untyped lambda calculus.

**Definition 4.2.1.** The *terms* of the simply typed lambda calculus are defined by

$$a, b \quad ::= \quad x \mid \lambda x.a \mid ab \mid * \mid \langle a, b\rangle \mid \texttt{let } * = a \texttt{ in } b \mid \texttt{let } \langle x, y\rangle = a \texttt{ in } b$$

where $x$ and $y$ are variables of the untyped lambda calculus.

Definition 4.2.1 extends the language of the untyped lambda calculus by adding the *constant* $*$ as well as two new term forming operations. The intended meaning of these new terms is as follows.

- $\langle a, b\rangle$ is the pair of $a$ and $b$.

- $*$ is the empty pair, i.e., the 0-ary version of $\langle a, b\rangle$.

- $\texttt{let}\ \langle x, y\rangle = a\ \texttt{in}\ b$ is a term that will reduce $a$ and in case $a \twoheadrightarrow \langle b_1, b_2\rangle$ will assign $b_1$ to $x$ and $b_2$ to $y$.

- $\texttt{let}\ * = a\ \texttt{in}\ b$ is the nullary version of $\texttt{let}\ \langle x, y\rangle = a\ \texttt{in}\ b$.

We extend the notion of free-variables of a term to account for the new term forming operations.

**Definition 4.2.2.** The set of *free variables* of a term $a$ of the simply typed lambda calculus, written $\mathrm{FV}(a)$, is defined as

- $\mathrm{FV}(x) = \{x\}$,

- $\mathrm{FV}(ab) = \mathrm{FV}(a) \cup \mathrm{FV}(b)$,

- $\mathrm{FV}(\lambda x.a) = \mathrm{FV}(a) \setminus \{x\}$,

- $\mathrm{FV}(*) = \emptyset$,

- $\mathrm{FV}(\langle a, b\rangle) = \mathrm{FV}(a) \cup \mathrm{FV}(b)$,

- $\mathrm{FV}(\texttt{let}\ * = a\ \texttt{in}\ b) = \mathrm{FV}(a) \cup \mathrm{FV}(b)$, and

- $\mathrm{FV}(\texttt{let}\ \langle x, y\rangle = a\ \texttt{in}\ b) = \mathrm{FV}(a) \cup (\mathrm{FV}(b) \setminus \{x, y\})$.

The notions of $\alpha$-equivalence and capture-avoiding substitution can be extended to the setting of the simply typed lambda calculus. Note that in $\texttt{let}\ \langle x, y\rangle = a\ \texttt{in}\ b$ the variables $x$ and $y$ are bound in $b$ (but not in $a$) so that $\texttt{let}$ is a binder. As in the untyped case, we say of a term $a$ such that $\mathrm{FV}(a) = \emptyset$ that it is closed.

### 4.2.2 Operational semantics

To account for the new term forming operations of our extended language, we introduce additional reduction rules.

$$\frac{}{\texttt{let}\ * = *\ \texttt{in}\ a \to a} \qquad \frac{}{\texttt{let}\ \langle x, y\rangle = \langle b, c\rangle\ \texttt{in}\ a \to a[b/x, c/y]}$$

$$\frac{b \to b'}{\langle a, b\rangle \to \langle a, b'\rangle} \qquad \frac{a \to a'}{\langle a, b\rangle \to \langle a', b\rangle}$$

$$\frac{a \to a'}{\texttt{let}\ * = a\ \texttt{in}\ b \to \texttt{let}\ * = a'\ \texttt{in}\ b}$$

$$\frac{a \to a'}{\texttt{let}\ \langle x, y\rangle = a\ \texttt{in}\ b \to \texttt{let}\ \langle x, y\rangle = a'\ \texttt{in}\ b}$$

$$\frac{b \to b'}{\texttt{let}\ * = a\ \texttt{in}\ b \to \texttt{let}\ * = a\ \texttt{in}\ b'}$$

$$\frac{b \to b'}{\texttt{let}\ \langle x, y\rangle = a\ \texttt{in}\ b \to \texttt{let}\ \langle x, y\rangle = a\ \texttt{in}\ b'}$$

Figure 4.1: Additional reduction rules for the simply typed lambda calculus.

**Definition 4.2.3.** The *one-step $\beta$-reduction*, written →, is defined on the terms of the simply typed lambda calculus by the rules of Definition 4.1.8 as well as those given in Figure 4.1. The *$\beta$-reduction*, written ↠, is the reflexive and transitive closure of →.

### 4.2.3 Types

**Definition 4.2.4.** The *types* of the simply typed lambda calculus are defined by

$$A, B \quad ::= \quad X \ \big|\ (A \times B) \ \big|\ 1 \ \big|\ (A \to B)$$

where $X$ comes from a set $\mathcal{T}$ of *basic types*.

It can be useful to think of types as sets of terms. Under this interpretation, we have

- $(A \times B)$ is the set of pairs,
- 1 is the set containing the unique empty tuple, and
- $(A \to B)$ is the set of functions from $A$ to $B$.

We now explain how to use types to restrict the formation of terms.

$$\frac{}{\Gamma, x : A \vdash x : A}\ (ax)$$

$$\frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x.b : A \to B}\ (\lambda) \qquad \frac{\Gamma \vdash c : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash ca : B}\ (app)$$

$$\frac{}{\Gamma \vdash * : 1}\ (*_i) \qquad \frac{\Gamma \vdash b : 1 \quad \Gamma \vdash a : A}{\Gamma \vdash \texttt{let}\ * = b\ \texttt{in}\ a : A}\ (*_e)$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \langle a, b \rangle : A \times B}\ (\times_i) \qquad \frac{\Gamma \vdash b : (B_1 \times B_2) \quad \Gamma, x : B_1, y : B_2 \vdash a : A}{\Gamma \vdash \texttt{let}\ \langle x, y \rangle = b\ \texttt{in}\ a : A}\ (\times_e)$$

Figure 4.2: Typing rules for the simply typed lambda calculus.

**Definition 4.2.5.** A *typing context* is a finite set $\{x_1 : A_1, \dots, x_n : A_n\}$ of pairs of a variable and a type, such that no variable occurs more than once. The expressions of the form $x : A$ in a typing context are called *type declarations*.

**Definition 4.2.6.** A *typing judgment* is an expression of the form

$$\Gamma \vdash a : A$$

where $\Gamma$ is a typing context, $a$ is a term, and $A$ is a type.

**Definition 4.2.7.** A typing judgment is *valid* if it can be inferred from the rules given in Figure 4.2.

If $a$ is a term, one shows that $\Gamma \vdash a : A$ is valid by exhibiting a *typing derivation*. If such a derivation exists, we say that $a$ is *well-typed of type $A$*, or sometimes simply *well-typed*. For example, below are two typing derivations, establishing that both $\lambda x.x$ and $\lambda xy.x$ are well-typed.

$$\frac{\frac{}{x : X \vdash x : X}}{\vdash \lambda x.x : X \to X} \qquad \frac{\frac{\frac{}{x : X, y : Y \vdash x : X}}{x : X \vdash \lambda y.x : Y \to X}}{\vdash \lambda xy.x : X \to (Y \to X)}$$

The term $xx$, however, is not well-typed. Indeed, suppose a typing derivation $\pi$ of $\Gamma \vdash xx : B$ exists. Then the last rule of $\pi$ must be the $(app)$ rule, since this rule is the only one allowing the construction of an application. Moreover, the only rule permitting the introduction of a variable is the $(ax)$ rule. The typing derivation $\pi$

must therefore be the following.

$$\frac{\overline{\Gamma, x : A \to B \vdash x : A \to B} \quad \overline{\Gamma, x : A \vdash x : A}}{\Gamma \vdash xx : B} \, .$$

But there are no types $A$ and $B$ such that $A = A \to B$. Hence there is no typing derivation of $\Gamma \vdash xx : B$.

### 4.2.4 Properties of the type system

The type system of the simply typed lambda calculus restricts the construction of terms in order to syntactically rule out "ill-behaved" terms. To verify that the type system achieves this intended goal, we need to prove that all well-typed terms "behave well". This intuitive idea is captured by establishing the *type safety* of the language. Following [17], we consider that a language is type safe if it enjoys the *subject reduction* and *progress* properties. The latter property relies on a notion of *value*. These values are a particular set of distinguished normal forms. The definition of value varies from language to language. For now, we can take the set of values to consist of all normal forms. Later, when we consider specific languages, this definition will be adjusted and explicitly stated.

**Subject reduction:** This property guarantees that the type of a term is stable under reduction. As a corollary, it also shows that if a term is well-typed, then it never reduces to an ill-typed term.

**Progress:** This property shows that a well-typed closed term is either a value or admits further reductions.

The simply typed lambda calculus is type safe as it enjoys both of the above properties.

**Proposition 4.2.8** (Subject reduction). *If* $\Gamma \vdash a : A$ *and* $a \to a'$*, then* $\Gamma \vdash a' : A$.

**Proposition 4.2.9** (Progress). *If* $\vdash a : A$*, then either* $a$ *is a value or there exists* $a'$ *such that* $a \to a'$*.*

The simply typed lambda calculus also enjoys a property known as *strong normalization.*

**Definition 4.2.10.** A term $a$ is *weakly normalizing* if there exists a finite sequence of reductions $a \to \ldots \to b$ where $b$ is in normal form, and *strongly normalizing* if every sequence of reductions starting from $a$ is finite.

Note that any strongly normalizing term is also weakly normalizing. Variables are examples of strongly normalizing terms. As an example of a term that is neither strongly nor weakly normalizing, consider $\Omega\Omega$, where $\Omega$ is the term $\lambda x.xx$. $\Omega\Omega$ is neither weakly nor strongly normalizing since we have

$$\Omega\Omega = (\lambda x.xx)\lambda x.xx \to xx[\lambda x.xx/x] = (\lambda x.xx)\lambda x.xx = \Omega\Omega.$$

As an example of a term that is weakly but not strongly normalizing, consider $(\lambda z.y)(\Omega\Omega)$.

Since $\Omega$ contains $xx$, we know that $\Omega\Omega$ is not well-typed. In contrast, the well-typed terms we have encountered so far, $\lambda x.x$ and $\lambda xy.x$, are both strongly normalizing. In fact, the simply typed lambda calculus has the property that *all* well-typed terms are strongly normalizable.

**Proposition 4.2.11** (Strong normalization)**.** *If* $\vdash a : A$ *is a valid typing judgment, then* $a$ *is strongly normalizing.*

## 4.3 Linearity

We now sketch a version of Girard's *intuitionistic linear logic* ([23]). As we shall see in the next section the use of linear logic in the context of quantum computation is motivated by the no-cloning property of quantum information.

### 4.3.1 Contraction, weakening, and strict linearity

Informally, a variable is used *linearly* if it is used exactly once. In the simply typed lambda calculus, variables can be used *non-linearly.* As a first example, consider the following typing derivation.

$$\dfrac{\overline{x : A \vdash x : A} \quad \overline{x : A \vdash x : A}}{x : A \vdash \langle x, x\rangle : A \times A}$$

In the above derivation the variable $x$ is used non-linearly, in the sense that only a single occurrence of $x$ in the context is required to construct the pair $\langle x, x\rangle$ in which $x$ occurs twice. This is possible because the two occurrences of the declaration $x : A$ in the leaves of the typing derivation were implicitly *contracted* by the application of the $(\times_i)$ rule. As another example, note that the typing judgement $x : A, y : B \vdash y : B$ is valid by the $(ax)$ rule. In this case, the variable $x$ is handled non-linearly because it appears in the context but is not used at all. This second kind of non-linearity is due to the implicit *weakening* of the context in the $(ax)$ rule.

It is possible to modify the typing rules of the simply typed lambda calculus to force variables to be used in a strictly linear fashion. To obtain such a system, we can replace the $(ax)$ and $(*_i)$ rules with

$$\frac{}{x : A \vdash x : A} \quad \text{and} \quad \frac{}{\vdash * : 1}\,.$$

In the above rules, the typing contexts are minimal, which guarantees that no implicit weakening can occur. To forbid implicit contractions we need to ensure that contexts are not merged but juxtaposed in binary rules. This can be achieved in the case of the $(\times_i)$ rule as follows.

$$\frac{\Gamma_1 \vdash a : A \quad \Gamma_2 \vdash b : B}{\Gamma_1, \Gamma_2 \vdash \langle a, b\rangle : A \times B}$$

The above rule carries the side condition that the contexts $\Gamma_1$ and $\Gamma_2$ are distinct, so that the notation $\Gamma_1, \Gamma_2$ denotes the disjoint union of the two contexts.

If contexts are removed from all nullary rules and juxtaposed rather than merged in all binary rules, then we obtain a strictly linear system. In this system, a variable occurs in the context of a valid typing judgement if and only if it appears exactly once in the term being typed.

### 4.3.2 Reintroducing non-linearity

The restrictions imposed by the strictly linear type system sketched above are very strong. Our interest in Girard's linear logic is the fact that it allows us to reintroduce a controlled form of non-linearity. The idea is to use a modality called *bang* and denoted ! to identify the variables that can be used non-linearly. To this end, we extend the grammar of types as follows.

$$A, B \quad ::= \quad X \mid 1 \mid (A \times B) \mid (A \to B) \mid\, !A$$

The new type $!A$ consists of all the elements of the type $A$ that can be used non-linearly. One can think of the elements of type $!A$ as those elements of $A$ that have the property of being *reusable* or *duplicable*.

We can modify the typing rules to account for this new modality. For example, the $(\times_i)$ rule becomes

$$\frac{!\Delta, \Gamma_1 \vdash a : A \quad !\Delta, \Gamma_2 \vdash b : B}{!\Delta, \Gamma_1, \Gamma_2 \vdash \langle a, b\rangle : A \times B} .$$

where the context $!\Delta$ denotes a set of declarations of the form $x_1 : !A_1, \ldots x_n : !A_n$. In this new rule, the contracted part of the context consists exclusively of declarations of the form $x : !A$. This ensures that the only variables that are used non-linearly are the ones of a non-linear type.

It should be possible to use a duplicable variable only once. In other words, if a variable $x$ is declared of type $!A$, it should also have type $A$. One way to achieve this is to equip the type system with a *subtyping relation*, denoted $<:$, satisfying $!A <: A$ for every type $A$.

## 4.4 The quantum lambda calculus

Various lambda calculi for quantum computation have appeared in the literature (e.g., [63], [3]). Here, we focus on the *quantum lambda calculus* (see [64], [61], or [58]) as it is the main inspiration for the Proto-Quipper language defined and studied in chapters 8 and 9.

The quantum lambda calculus is based on the QRAM model of quantum computation described in Section 2.2.4. To embody the QRAM model, the reduction relation of the quantum lambda calculus is defined on *closures*. These closures are triples $[Q, L, a]$ where $Q$ is a unit vector in $\bigotimes_{i=1}^{n} \mathbb{C}^2$ for some integer $n$, $L$ is a list of $n$ distinct term variables, and $a$ is a term of the quantum lambda calculus.

The vector $Q$ represents the state of a system of $n$ qubits held in some hypothetical quantum device. In a well-formed closure, the free variables of $a$ are required to form a subset of $L$ and the list $L$ is interpreted as a link between the variables of $a$ and the qubits of $Q$. This way, the qubits whose state is described by $Q$ become accessible to the operations of the quantum lambda calculus.

### 4.4.1 Terms

**Definition 4.4.1.** The *terms* of the quantum lambda calculus are defined by

$$\begin{array}{rcl} a, b, c & ::= & x \;\big|\; u \;\big|\; \lambda x.a \;\big|\; ab \;\big|\; \langle a, b\rangle \;\big|\; * \;\big| \\ & & \texttt{let}\ \langle x, y\rangle = a\ \texttt{in}\ b \;\big|\; \texttt{let rec}\ x\ y = b\ \texttt{in}\ c \;\big| \\ & & \texttt{injl}(a) \;\big|\; \texttt{injr}(a) \;\big|\; \texttt{match}\ a\ \texttt{with}\ (x \mapsto b \mid y \mapsto c) \end{array}$$

where $u$ comes from a set $U$ of *quantum constants* and $x, y$ come from a countable set $\mathcal{V}$ of *variables*.

The meaning of most terms is intended to be the standard one, as described in the previous sections. The term $\texttt{let rec}\ x\ y = b\ \texttt{in}\ c$ is a recursion operator. The terms $\texttt{injl}(a)$ and $\texttt{injr}(a)$ denote the left and right inclusion in a disjoint union respectively. The term $\texttt{match}\ a\ \texttt{with}\ (x \mapsto b \mid y \mapsto c)$ denotes a case distinction depending on $a$. The terms $\texttt{injl}(*)$ and $\texttt{injr}(*)$ form a two element set on which one can perform a case distinction. The classical *bits* are defined as the elements of this set, with $0 = \texttt{injr}(*)$ and $1 = \texttt{injl}(*)$.

The set $U$ contains syntactical representatives of certain operations that can be executed by the quantum device. In particular, $U$ is assumed to contain the constants `new` and `meas`, whose intended interpretation is as follows.

- The term `new` represents an initialization function. It inputs a bit (i.e., one of 0 or 1 as defined above) and produces a qubit in the corresponding classical state (i.e., $|1\rangle$ or $|0\rangle$ respectively).
- The term `meas` represents a measurement function. It inputs a qubit and measures it in the computational basis, returning the corresponding bit.

Assume that $U$ contains a constant $H$ representing the Hadamard gate and consider the term `coin` defined as

$$\texttt{coin} = \lambda * .\, \texttt{meas}(H(\texttt{new}\, 0)).$$

This term represents a “fair coin”. When applied to any argument, it will prepare a qubit in the state $|0\rangle$ and apply a Hadamard gate to it. This results in the superposition

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}}.$$

The qubit is then measured, which results in 0 or 1 with equal probability.

### 4.4.2 Operational semantics

In the quantum lambda calculus, one must choose a *reduction strategy.* To see why this is the case, assume that bplus  is a term from the quantum lambda calculus representing addition modulo 2 and consider the following term

$$a = (\lambda x.\texttt{bplus}\ \langle x, x\rangle)(\texttt{coin}\ *).$$

The term $a$ has two redexes. If the outer redex is reduced first, we obtain the term $\texttt{bplus}\ \langle(\texttt{coin}\ *), (\texttt{coin}\ *)\rangle$, which will reduce to 0 or 1 with equal probability. However, if we evaluate $(\texttt{coin}\ *)$ first, then $a$ reduces to

$$(\lambda x.\texttt{bplus}\ \langle x, x\rangle)0 \quad \text{or} \quad (\lambda x.\texttt{bplus}\ \langle x, x\rangle)1$$

with equal probability. Either way, the final result of computation will be 0. This example shows that confluence fails in the quantum lambda calculus, forcing us to choose an order of evaluation. In the quantum lambda calculus, evaluation of terms follows a *call-by-value* reduction strategy. In particular, this means that when evaluating an application we reduce the argument before applying the function. To determine when a term is reduced, we define a notion of *value.*

**Definition 4.4.2.** The *values* of the quantum lambda calculus are defined by

$$v, w \quad ::= \quad x \mid u \mid * \mid \langle v, w\rangle \mid \lambda x.a \mid \texttt{injr}(v) \mid \texttt{injl}(v).$$

**Definition 4.4.3.** A *closure* is a triple $[Q, L, a]$ where

- $Q$ is a normalized vector of $\mathbb{C}^{2^n}$ for some $n \geqslant 0$ called a *quantum array,*
- $L$ is a list of $n$ distinct term variables, and
- $a$ is a term whose free variables appear in $L$.

The closure $[Q, L, a]$ is a *value* if $a$ is a value.

**Definition 4.4.4.** The *one-step β-reduction*, written $\to_p$, is defined on closures by the rules given in Figure 4.3. The notation $[Q, L, a] \to_p [Q', L', a']$ means that the reduction takes place with probability $p$.

$$\frac{[Q,L,a] \to_p [Q',L',a']}{[Q,L,av] \to_p [Q',L',a'v]} \qquad \frac{[Q,L,b] \to_p [Q',L',b']}{[Q,L,ab] \to_p [Q',L',ab']}$$

$$\frac{[Q,L,b] \to_p [Q',L',b']}{[Q,L,\langle a,b\rangle] \to_p [Q',L',\langle a,b'\rangle]} \qquad \frac{[Q,L,a] \to_p [Q',L',a']}{[Q,L,\langle a,v\rangle] \to_p [Q',L',\langle a',v\rangle]}$$

$$\frac{[Q,L,a] \to_p [Q',L',a']}{[Q,L,\texttt{injl}(a)] \to_p [Q',L',\texttt{injl}(a')]} \qquad \frac{[Q,L,a] \to_p [Q',L',a']}{[Q,L,\texttt{injr}(a)] \to_p [Q',L',\texttt{injr}(a')]}$$

$$\frac{[Q,L,a] \to_p [Q',L',a']}{[Q,L,\texttt{let}\ \langle x,y\rangle = a\ \texttt{in}\ b] \to_p [Q',L',\texttt{let}\ \langle x,y\rangle = a'\ \texttt{in}\ b]}$$

$$\frac{[Q,L,a] \to_p [Q',L',a']}{[Q,L,\texttt{match}\ a\ \texttt{with}\ (x \mapsto b \mid y \mapsto c)] \to_p [Q',L',\texttt{match}\ a'\ \texttt{with}\ (x \mapsto b \mid y \mapsto c)]}$$

$$[Q,L,(\lambda x.a)v] \to_1 [Q,L,a[v/x]]$$

$$[Q,L,\texttt{let}\ * = *\ \texttt{in}\ a] \to_1 [Q,L,a]$$

$$[Q,L,\texttt{let}\ \langle x,y\rangle = \langle v,w\rangle\ \texttt{in}\ a] \to_1 [Q,L,a[v/x,w/y]]$$

$$[Q,L,\texttt{match}\ \texttt{injl}(v)\ \texttt{with}\ (x \mapsto b \mid y \mapsto c)] \to_1 [Q,L,b[v/x]]$$

$$[Q,L,\texttt{match}\ \texttt{injr}(v)\ \texttt{with}\ (x \mapsto b \mid y \mapsto c)] \to_1 [Q,L,c[v/y]]$$

$$[Q,L,\texttt{let rec}\ x\ y = b\ \texttt{in}\ c] \to_1 [Q,L,c[(\lambda y.\texttt{let rec}\ x\ y = b\ \texttt{in}\ b)/x]]$$

$$[Q,L,u\langle x_{j_1},\ldots,x_{j_n}\rangle] \to_1 [Q',L,\langle x_{j_1},\ldots,x_{j_n}\rangle]$$

$$[\alpha|Q_0\rangle + \beta|Q_1\rangle, L, \texttt{meas}(x_i)] \to_{|\alpha|^2} [|Q_0\rangle, L, 0]$$

$$[\alpha|Q_0\rangle + \beta|Q_1\rangle, L, \texttt{meas}(x_i)] \to_{|\beta|^2} [|Q_1\rangle, L, 1]$$

$$[Q, |x_1 \ldots x_n\rangle, \texttt{new}(0)] \to_1 [Q \otimes |0\rangle, |x_1 \ldots x_{n+1}\rangle, x_{n+1}]$$

$$[Q, |x_1 \ldots x_n\rangle, \texttt{new}(1)] \to_1 [Q \otimes |1\rangle, |x_1 \ldots x_{n+1}\rangle, x_{n+1}]$$

Figure 4.3: Reduction rules for the quantum lambda calculus.

The rules are separated in three groups. The first group contains the *congruence rules*. In particular, the rules for the reduction of an application can be seen to define a call-by-value reduction strategy. The second group of rules contains the *classical rules*. These rules define the reduction of redexes that do not involve any of the constants from the set $U$. The last group of rules contains the *quantum rules*. These rules define the interaction between the classical device and the quantum device. In the first quantum rule, we have $Q' = u(Q)$. This rule corresponds to the application of the unitary $u$ to the relevant qubits. Note that the only probabilistic reduction step is the one corresponding to measurement.

The chosen reduction strategy guarantees that, at every step of a reduction, only one rule applies. Hence, unlike the untyped lambda calculus, the quantum lambda calculus is *deterministic*.

**Proposition 4.4.5.** *If* $[Q, L, a]$ *is a closure, then at most one reduction rule applies to it.*

### 4.4.3 Types

**Definition 4.4.6.** The *types* of the quantum lambda calculus are defined by

$$A, B \quad ::= \quad \mathbf{qubit} \;\big|\; 1 \;\big|\; !A \;\big|\; A \otimes B \;\big|\; A \oplus B \;\big|\; A \multimap B.$$

The type system of the quantum lambda calculus is based on intuitionistic linear logic as sketched in Section 4.3. The notation is adopted from linear logic, with $A \otimes B$ for the type of pairs, $A \oplus B$ for the type of sums, and $A \multimap B$ for the type of functions. The type **qubit** represents the set of all 1-qubit states. As in Section 4.3, the type $!A$ can be understood as the subset of $A$ consisting of values that have the additional property of being *duplicable* or *reusable*. We will sometimes write $!^n A$, with $n \in \mathbb{N}$, to mean

$$\underbrace{!\ldots!}_{n} A.$$

Similarly, we sometimes write $A^{\otimes n}$ to mean

$$\underbrace{A \otimes \ldots \otimes A}_{n}.$$

$$\frac{}{\mathbf{qubit} <: \mathbf{qubit}} \qquad \frac{}{1 <: 1}$$

$$\frac{A_1 <: B_1 \quad A_2 <: B_2}{(A_1 \otimes A_2) <: (B_1 \otimes B_2)} \quad \frac{A_1 <: B_1 \quad A_2 <: B_2}{(A_1 \oplus A_2) <: (B_1 \oplus B_2)} \quad \frac{A_2 <: A_1 \quad B_1 <: B_2}{(A_1 \multimap B_1) <: (A_2 \multimap B_2)}$$

$$\frac{A <: B \quad (n = 0 \Rightarrow m = 0)}{!^n A <: !^m B}$$

Figure 4.4: Subtyping rules for the quantum lambda calculus.

We also write **bit** for the type $1 \oplus 1$.

The fact that a term is reusable should not prevent us from using it exactly once. Intuitively, this should imply that if $!A$ is a valid type for a given term, then $A$ should also be a valid type for it. To capture this idea, we use a subtyping relation on types.

**Definition 4.4.7.** The *subtyping relation* $<:$ is the smallest relation on types satisfying the rules given in Figure 4.4.

**Proposition 4.4.8.** *The subtyping relation is reflexive and transitive.*

To define the type system of the quantum lambda calculus, we first introduce axioms for the elements of the set $U$.

**Definition 4.4.9.** We introduce a type for the constants of $U$. For `new` and `meas` we set

$$A_{\texttt{new}} = \mathbf{bit} \multimap \mathbf{qubit}, \quad A_{\texttt{meas}} = \mathbf{qubit} \multimap !\mathbf{bit},$$

and for the remaining elements $v \in U$ we set

$$A_v = \mathbf{qubit}^{\otimes n} \multimap \mathbf{qubit}^{\otimes n}.$$

**Definition 4.4.10.** A typing judgment of the quantum lambda calculus is *valid* if it can be inferred from the rules given in Figure 4.5.

There are two rules for the construction of a lambda abstraction. The rule $(\lambda_1)$ is similar to the $(\lambda)$ rule of the simply typed lambda calculus. However, the produced function is not duplicable. In contrast, the rule $(\lambda_2)$ produces a duplicable function. The main difference is that in the $(\lambda_2)$ rule, the free variables that appear in $b$ must

$$\frac{A <: B}{\Gamma, x : A \vdash x : B}\ (ax_1) \qquad \frac{!A_u <: B}{\Gamma \vdash u : B}\ (ax_2)$$

$$\frac{\Gamma \vdash a : !^n A}{\Gamma \vdash \texttt{injl}(a) : !^n(A \oplus B)}\ (\oplus_{i_1}) \qquad \frac{\Gamma \vdash b : !^n B}{\Gamma \vdash \texttt{injr}(b) : !^n(A \oplus B)}\ (\oplus_{i_2})$$

$$\frac{\begin{array}{r} \Gamma_2, !\Delta, y : !^n A \vdash c : C \\ \Gamma_1, !\Delta \vdash a : !^n(A \oplus B) \quad \Gamma_2, !\Delta, x : !^n A \vdash b : C \end{array}}{\Gamma_1, \Gamma_2, !\Delta \vdash \texttt{match}\ a\ \texttt{with}\ (x \mapsto b \mid y \mapsto c) : C}\ (\oplus_e)$$

$$\frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x.b : A \multimap B}\ (\lambda_1) \qquad \frac{\Gamma, !\Delta, x : A \vdash b : B \quad \mathrm{FV}(b) \cap |\Gamma| = \emptyset}{\Gamma, !\Delta \vdash \lambda x.b : !^{n+1}(A \multimap B)}\ (\lambda_2)$$

$$\frac{\Gamma_1, !\Delta \vdash c : A \multimap B \quad \Gamma_2, !\Delta \vdash a : A}{\Gamma_1, \Gamma_2, !\Delta \vdash ca : B}\ (app)$$

$$\frac{}{\Gamma \vdash * : !^n 1}\ (*_i)$$

$$\frac{\Gamma_1, !\Delta; Q_1 \vdash a : !^n A \quad \Gamma_2, !\Delta; Q_2 \vdash b : !^n B}{\Gamma_1, \Gamma_2, !\Delta \vdash \langle a, b \rangle : !^n(A \otimes B)}\ (\otimes_i)$$

$$\frac{\Gamma_1, !\Delta \vdash b : !^n(B_1 \otimes B_2) \quad \Gamma_2, !\Delta, x : !^n B_1, y : !^n B_2 \vdash a : A}{\Gamma_1, \Gamma_2, !\Delta \vdash \texttt{let}\ \langle x, y \rangle = b\ \texttt{in}\ a : A}\ (\otimes_e)$$

$$\frac{!\Delta, x : !(A \multimap B), y : A \vdash b : B \quad \Gamma, !\Delta, x : !(A \multimap B) \vdash c : C}{\Gamma, !\Delta \vdash \texttt{let rec}\ x\ y = b\ \texttt{in}\ c : C}\ (rec)$$

Figure 4.5: Typing rules for the quantum lambda calculus.

all be of a duplicable type. This prevents $b$ from having any embedded quantum data, which could not be cloned. Note that the type system prevents us from assigning the type **qubit** $\multimap$ **qubit** $\otimes$ **qubit** to the term $\lambda x.\langle x, x \rangle$.

**Definition 4.4.11.** A *typed closure* is an expression of the form

$$[Q, L, a] : A,$$

where $[Q, L, a]$ is a closure and $A$ is a type. It is *valid* if

$$x_1 : \mathbf{qubit}, \ldots, x_n : \mathbf{qubit} \vdash a : A$$

is a valid typing judgement, with $L = |x_1, \ldots, x_n\rangle$.

The quantum lambda calculus is a type safe language, in the sense that it enjoys the subject reduction and progress properties.

**Proposition 4.4.12.** *If $[Q, L, a] : A$ is a valid typed closure and*

$$[Q, L, a] \to_p [Q', L', a'],$$

*then $[Q', L', a'] : A$ is a valid typed closure.*

**Proposition 4.4.13.** *Let $[Q, L, a]$ be a valid typed closure of type $A$. Then either $[Q, L, a]$ is a value, or there is a valid typed closure $[Q', L', a']$ such that $[Q, L, a] \to_p [Q', L', a']$. Moreover, the total probability of all possible one-step reductions from $[Q, L, a]$ is 1.*

# Chapter 5

# Grid problems

In this chapter, we present an efficient method to solve a type of lattice point enumeration problem which we call a *grid problem*. As a first approximation, a grid problem can be thought of as follows: given a discrete subset $L \subseteq \mathbb{R}^2$ (such as a lattice), which we call the *grid*, and a bounded convex subset $A \subseteq \mathbb{R}^2$ with non-empty interior, enumerate all the points $u \in A \cap L$. Specifically, we will be interested in grid problems for which $L$ is a subset of $\mathbb{Z}[i]$ or of $\mathbb{Z}[\omega]$, as defined in Chapter 3. We refer to the first kind of problem as a *grid problem over* $\mathbb{Z}[i]$ and to the second kind of problem as a *grid problem over* $\mathbb{Z}[\omega]$. As we shall see in chapters 6 and 7, these problems have applications in quantum computation. However, they are treated here independently of any quantum considerations. The results contained in this chapter first appeared in [54] and [56].

We note that the method presented here is not the only method for solving grid problems. Alternatively, grid problems over $\mathbb{Z}[i]$ or $\mathbb{Z}[\omega]$ can be reduced to so-called *integer programming problems* in some fixed dimension which can be efficiently solved using the techniques pioneered by Lenstra in [45]. Nevertheless, we believe our method is novel and interesting.

Even though grid problems over $\mathbb{Z}[i]$ are significantly simpler than grid problems over $\mathbb{Z}[\omega]$, the overall method remains the same. For this reason, the case of $\mathbb{Z}[i]$, which is treated first, is used as an introduction to the case of $\mathbb{Z}[\omega]$, which is treated second.

Let $R$ be one of $\mathbb{Z}[i]$ or $\mathbb{Z}[\omega]$. As in Chapter 3, we quantify the complexity of our methods by estimating the number of arithmetic operations required to produce an element $u \in A \cap L$, for $L \subseteq R$. Our algorithms will input bounded convex subsets $A \subseteq \mathbb{R}^2$ (as well as closed intervals $[x_0, x_1] \subseteq \mathbb{R}$). If we were to give a rigorous complexity-theoretic account, we should indicate what it means for a subset $A$ of $\mathbb{R}^2$ to be "given" as the input to an algorithm. The details of this do not matter much.

For our purposes, it will suffice to assume that a convex set is given along with the following information.

- A convex polygon enclosing $A$, say with rational vertices, and such that the area of the polygon exceeds that of $A$ by at most a fixed constant factor;
- a method to decide, for any given point of $R$, whether it is in $A$ or not; and
- a method to compute the intersection of $A$ with any straight line in $R$. More precisely, given any straight line parameterized as $L(t) = p + tq$, with $p, q \in R$, we can effectively determine the interval $\{t \mid L(t) \in A\}$ in the sense of the above.

## 5.1 Grid problems over $\mathbb{Z}[i]$

Recall from Chapter 3 that the elements of $\mathbb{Z}[i]$ are of the form $a + bi$, with $a$ and $b$ in $\mathbb{Z}$. We can therefore identify $\mathbb{Z}[i]$ with the set $\mathbb{Z}^2 \subseteq \mathbb{R}^2$. When viewed in this way, we refer to $\mathbb{Z}[i]$ as the *grid* and to elements $u \in \mathbb{Z}[i]$ as *grid points*.

**Problem 5.1.1** (Grid problem over $\mathbb{Z}[i]$)**.** Given a bounded convex subset $A$ of $\mathbb{R}^2$ with non-empty interior, enumerate all the points $u \in A \cap \mathbb{Z}[i]$.

A point $u \in A \cap \mathbb{Z}[i]$ is called a *solution* to the grid problem over $\mathbb{Z}[i]$ for $A$. Figure 5.1 (a) illustrates a grid problem for which $A$ is a disk centered at the origin. The grid is shown as black dots and the set $A$ is shown in red.

### 5.1.1 Upright rectangles

We first consider grid problems over $\mathbb{Z}[i]$ for which $A$ is an *upright rectangle*, i.e., of the form $[x_1, x_2] \times [y_1, y_2]$. These instances are easily solved, as they can be reduced to a problem in a lower dimension. Indeed, it suffices to independently solve the grid problem in the $x$-axis (i.e., by enumerating the integers in $[x_1, x_2]$) and in the $y$-axis (i.e. by enumerating the integers in interval $[y_1, y_2]$), as illustrated in Figure 5.1 (b).

**Proposition 5.1.2.** *Let $A$ be an upright rectangle. Then there is an algorithm which enumerates all the solutions to the grid problem over $\mathbb{Z}[i]$ for $A$. Moreover, the algorithm requires only a constant number of arithmetic operations per solution produced.*

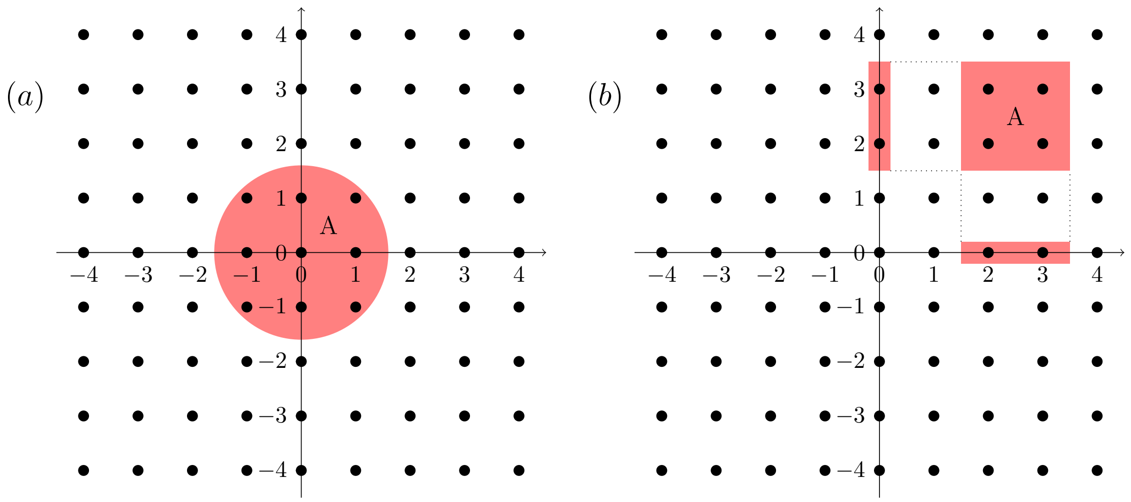


Figure 5.1: (a) A grid problem over $\mathbb{Z}[i]$ for which $A$ is a disk of radius 1.5 centered at the origin. (b) A grid problem over $\mathbb{Z}[i]$ for which $A$ is an upright rectangle, together with the projections of $A$ along the $x$- and $y$-axes.

### 5.1.2 Upright sets

We now generalize the method of the previous subsection to convex sets that are *close* to upright rectangles in a suitable sense.

**Definition 5.1.3** (Uprightness)**.** Let $A$ be a bounded convex subset of $\mathbb{R}^2$ with non-empty interior. The bounding box of $A$, denoted $\mathrm{BBox}(A)$, is the smallest set of the form $[x_1, x_2] \times [y_1, y_2]$ that contains $A$. The *uprightness of* $A$, denoted $\mathrm{up}(A)$, is defined to be the ratio of the area of A to the area of its bounding box:

$$\mathrm{up}(A) = \frac{\mathrm{area}(A)}{\mathrm{area}(\mathrm{BBox}(A))}.$$

We say that $A$ is $M$-upright if $\mathrm{up}(A) \geqslant M$.

**Proposition 5.1.4.** *Let $A$ be an $M$-upright set. Then there exists an algorithm which enumerates all the solutions to the grid problem over $\mathbb{Z}[i]$ for $A$. Moreover, the algorithm requires $O(1/M)$ arithmetic operations per solution produced. In particular, when $M > 0$ is fixed, it requires only a constant number of operations per solution.*

*Proof.* By Proposition 5.1.2, we can efficiently enumerate the solutions of the grid problem for $\mathrm{BBox}(A)$. For each such candidate solution $u$, we only need to check

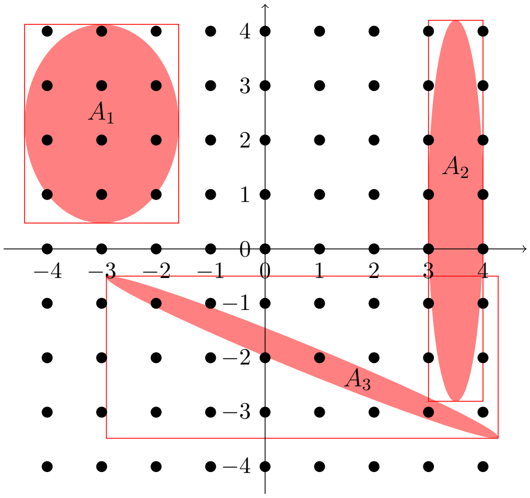


Figure 5.2: Grid problems over $\mathbb{Z}[i]$ for upright and non-upright sets.

whether $u$ is also a solution for $A$. To establish the efficiency of the algorithm, we need to ensure that the total number of solutions is not too small in relation to the total number of candidates produced. To see this, note that, with the exception of trivial cases, when the number of rows or columns is very small, $M$-uprightness and convexity ensure that the proportion of candidates $u$ that are solutions for $A$ is approximately $M : 1$. Therefore, the runtime per solution differs from that of Proposition 5.1.2 by at most a factor of $O(1/M)$. □

Figure 5.2 shows three different examples of grid problems over $\mathbb{Z}[i]$. The sets $A_i$ are again shown in red, for $i = 1, 2, 3$, and their bounding boxes are shown in outline. The typical case of an upright set is $A_1$. Here, a fixed proportion of grid points from the bounding box of $A_1$ are elements of $A_1$. The exceptional case of an upright set is $A_2$. Its bounding box spans only two columns of the grid. Therefore, although the bounding box contains many grid points, $A_2$ does not. However, this case is easily dealt with by solving the problem in a lower dimension for each of the grid columns separately. Finally, the set $A_3$ is not upright. In this case, Lemma 5.1.4 is not helpful, and a priori, it could be a difficult problem to find grid points in $A_3$.

### 5.1.3 Grid operators

The method of the previous subsection can be further generalized by using certain linear transformations to turn non-upright sets into upright sets. The linear transformations that are useful for this purpose are *special grid operators*.

**Definition 5.1.5** (Grid operator)**.** A *grid operator* is an integer matrix, or equivalently, a linear operator, that maps $\mathbb{Z}^2$ to itself. A grid operator $G$ is called *special* if it has determinant $\pm 1$, in which case $G^{-1}$ is also a grid operator.

*Remark* 5.1.6. If $A$ is a subset of $\mathbb{R}^2$ and $G$ is a grid operator, then $G(A)$, the direct image of $A$, is defined as usual by $G(A) = \{G(v) \ ; \ v \in A\}$.

*Remark* 5.1.7. The interest in special grid operators lies in the fact that $u$ is a solution to the grid problem over $\mathbb{Z}[i]$ for $A$ if and only if $G(u)$ is a solution to the grid problem over $\mathbb{Z}[i]$ for $G(A)$.

### 5.1.4 Ellipses

Combining the results of the previous two subsections, we know that the grid problem over $\mathbb{Z}[i]$ for $A$ can be solved efficiently provided that we can find a operator $G$ such that $G(A)$ is sufficiently upright. In this subsection, we show that if $A$ is an ellipse, then this can always be done.

**Definition 5.1.8** (Ellipse)**.** Let $D$ be a positive definite real $2 \times 2$-matrix with non-zero determinant, and let $p \in \mathbb{R}^2$ be a point. The *ellipse defined by $D$ and centered at $p$* is the set

$$E = \{u \in \mathbb{R}^2 \ ; \ (u-p)^\dagger D(u-p) \leqslant 1\}.$$

*Remark* 5.1.9. If $G$ is a grid operator and $E$ is an ellipse centered at the origin and defined by $D$, then $G(E)$ is an ellipse centered at the origin and defined by $(G^{-1})^\dagger D G^{-1}$.

The notion of uprightness introduced above was defined for an arbitrary bounded convex subset of $\mathbb{R}^2$. If the set in question is an ellipse, we can expand the definition of uprightness into an explicit expression.

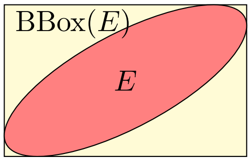
**Proposition 5.1.10.** *Let $E$ be the ellipse defined by $D$ and centered at $p$, with*

$$D = \begin{bmatrix} a & b \\ b & d \end{bmatrix}.$$

*Then* $\mathrm{up}(E) = \frac{\pi}{4}\sqrt{\frac{\det(D)}{ad}}$.

*Proof.* We can compute the area of $E$ and the area of its bounding box using $D$. Indeed, we have $\mathrm{area}(E) = \pi/\sqrt{\det(D)}$ and $\mathrm{area}(\mathrm{BBox}(E)) = 4\sqrt{ad}/\det(D)$. Substituting these in Definition 5.1.3, yields the desired expression for uprightness

$$\mathrm{up}(E) = \frac{\mathrm{area}(E)}{\mathrm{area}(\mathrm{BBox}(E))} = \frac{\pi}{4}\sqrt{\frac{\det(D)}{ad}}.$$



□

*Remark* 5.1.11. The uprightness of an ellipse is invariant under translation and scalar multiplication.

**Definition 5.1.12** (Skew)**.** The *skew* of a matrix is the product of its anti-diagonal entries. The *skew* of an ellipse defined by $D$ is the skew of $D$.

By Proposition 5.1.10, the skew of an ellipse is small if and only if its uprightness is large. Our strategy for increasing the uprightness will therefore be to reduce the skew.

**Proposition 5.1.13.** *Let $E$ be an ellipse. There exists a grid operator $G$ such that $G(E)$ is $1/2$-upright. Moreover, if $E$ is $M$-upright, then $G$ can be efficiently computed in $O(\log(1/M))$ arithmetic operations.*

*Proof.* Let $A$ and $B$ be the following special grid operators

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix},$$

and consider an ellipse $E$ defined by $D$ and centered at $p$. Since uprightness is invariant under translation and scaling, we may without loss of generality assume that $E$ is centered at the origin and that $D$ has determinant 1. Suppose moreover that the entries of $D$ are as follows

$$\begin{bmatrix} a & b \\ b & d \end{bmatrix}.$$

Note that $D$ can be written in this form because it is symmetric. We first show that there exists a grid operator $G$ such that $\texttt{Skew}(G(E)) \leqslant 1$. Indeed, assume that $\texttt{Skew}(E) = b^2 \geqslant 1$. In case $a \leqslant d$, choose $n$ such that $|na + b| \leqslant a/2$. Then we have:

$$A^{n\dagger} D A^n = \begin{bmatrix} \cdots & na+b \\ na+b & \cdots \end{bmatrix}.$$

Therefore, using Remark 5.1.9 with $G_1 = (A^n)^{-1}$, we have:

$$\begin{aligned} \texttt{Skew}(G_1(E)) = (na+b)^2 \leqslant \frac{a^2}{4} \leqslant \frac{ad}{4} &= \frac{1+b^2}{4} \\ &= \frac{1+\texttt{Skew}(E)}{4} \\ &\leqslant \frac{2\ \texttt{Skew}(E)}{4} = \frac{1}{2}\texttt{Skew}(E). \end{aligned}$$

Similarly, in case $d < a$, then choose $n$ such that $|nd+b| \leqslant d/2$. A similar calculation shows that in this case, with $G_1 = (B^n)^{-1}$, we get $\texttt{Skew}(G_1(E)) \leqslant \frac{1}{2}\texttt{Skew}(E)$. In both cases, the skew of $E$ is reduced by a factor of 2 or more. Applying this process repeatedly yields a sequence of operators $G_1, \ldots, G_m$ and letting $G = G_m \cdot \ldots \cdot G_1$ we find that $\texttt{Skew}(G(E)) \leqslant 1$.

Now let $D'$ be the matrix defining $G(E)$, with entries as follows:

$$D' = \begin{bmatrix} \alpha & \beta \\ \beta & \delta \end{bmatrix}.$$

Then $\texttt{Skew}(G(E)) \leqslant 1$ implies that $\beta^2 \leqslant 1$. Moreover, since $A$ and $B$ are special grid operators we have $\det(D') = \alpha\delta - \beta^2 = 1$. Using the expression from Proposition 5.1.10 for the uprightness of $G(E)$ we get the desired result:

$$\operatorname{up}(G(E)) = \frac{\pi}{4}\sqrt{\frac{\det(D')}{\alpha\delta}} = \frac{\pi}{4\sqrt{\alpha\delta}} = \frac{\pi}{4\sqrt{\beta^2+1}} \geqslant \frac{\pi}{4\sqrt{2}} \geqslant \frac{1}{2}.$$

Finally, to bound the number of arithmetic operations, note that each application of $G_j$ reduces the skew by at least a factor of 2. Therefore, the number $n$ of grid operators required satisfies $n \leqslant \log_2(\texttt{Skew}(E))$. Now note that since $D$ has determinant

1, we have

$$M \leqslant \mathrm{up}(E) = \frac{\pi}{4}\frac{1}{\sqrt{ad}} = \frac{\pi}{4\sqrt{b^2+1}}.$$

Therefore $\texttt{Skew}(E) = b^2 \leqslant (\pi^2/16M^2) - 1$, so that the computation of $G$ requires $O(\log(1/M))$ arithmetic operations. □

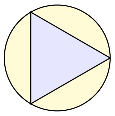
### 5.1.5 The enclosing ellipse of a bounded convex set

The final step in our solution of grid problems over $\mathbb{Z}[i]$ is to generalize Proposition 5.1.13 from ellipses to arbitrary bounded convex sets with non-empty interior. This can be done because every such set $A$ can be inscribed in an ellipse whose area is not much greater than that of $A$, as stated in the following proposition, which was proved in [56].

**Proposition 5.1.14.** *Let $A$ be a bounded convex subset of $\mathbb{R}^2$ with non-empty interior. Then there exists an ellipse $E$ such that $A \subseteq E$, and such that*

$$\mathrm{area}(E) \leqslant \frac{4\pi}{3\sqrt{3}}\,\mathrm{area}(A). \tag{5.1}$$

Note that $\frac{4\pi}{3\sqrt{3}} \approx 2.4184$. We remark that the bound in Proposition 5.1.14 is sharp; the bound is attained in case $A$ is an equilateral triangle. In this case, the enclosing ellipse is a circle, and the ratio of the areas is exactly $\frac{4\pi}{3\sqrt{3}}$.



### 5.1.6 General solution to grid problems over $\mathbb{Z}[i]$

We can now describe our algorithm to solve Problem 5.1.1.

**Proposition 5.1.15.** *There is an algorithm which, given a bounded convex subset $A$ of $\mathbb{R}^2$ with non-empty interior, enumerates all solutions of the grid problem over $\mathbb{Z}[i]$ for $A$. Moreover, if $A$ is $M$-upright, then the algorithm requires $O(\log(1/M))$ arithmetic operations overall, plus a constant number of arithmetic operations per solution produced.*

*Proof.* Given $A$, with an enclosing ellipse $A'$ whose area only exceeds that of $A$ by a fixed constant factor $N$, use Proposition 5.1.13 to find a grid operator $G$ such that $G(A')$ is 1/2-upright. Then, use Proposition 5.1.4 to enumerate the grid points of $G(A')$. For each grid point $u$ found, check whether it belongs to $G(A)$. This is the case if and only if $G^{-1}(u)$ is a solution to the grid problem over $\mathbb{Z}[i]$ for $A$. □

*Remark* 5.1.16. Note that the complexity of $O(\log(1/M))$ overall operations in Proposition 5.1.15 is exponentially better than the complexity of $O(1/M)$ per candidate we obtained in Proposition 5.1.4. This improvement is entirely due to the use of grid operators in Proposition 5.1.13.

### 5.1.7 Scaled grid problems over $\mathbb{Z}[i]$

If $A$ is a convex bounded subset of $\mathbb{R}^2$ with non-empty interior, then so is the set $rA$, for any non-zero real number $r$. Hence, by the results of the previous subsection, we can solve grid problems over $\mathbb{Z}[i]$ for $rA$ where $r$ is a non-zero scalar and $A$ is a bounded convex subset of $\mathbb{R}^2$ with non empty interior. We call such a problem a *scaled grid problem over $\mathbb{Z}[i]$ for $A$ and $r$.* In Chapter 6, we will be interested in solving a sequence of such scaled grid problems for specific values of $r$. The reasons behind our particular choice of a sequence will be detailed in Chapter 6.

Consider the set of real numbers of the form $\sqrt{2}^k\sqrt{5}^\ell$, with $k, \ell \in \mathbb{N}$ and $0 \leqslant k \leqslant 2$. Note that the set $\{\sqrt{2}^k\sqrt{5}^\ell\}_{k,\ell}$ is ordered as a subset of $\mathbb{R}$. In particular, if $\sqrt{2}^k\sqrt{5}^\ell \leqslant \sqrt{2}^{k'}\sqrt{5}^{\ell'}$, then $\ell \leqslant \ell'$. When we say that an algorithm "enumerates all solutions of the scaled grid problem over $\mathbb{Z}[i]$ for $A$ and $\sqrt{2}^k\sqrt{5}^\ell$ in order of increasing $\ell$", we mean that the algorithm first outputs all solutions for $\ell = 0$, then for $\ell = 1$, etc.

**Proposition 5.1.17.** *There is an algorithm which, given a bounded convex subset $A$ of $\mathbb{R}^2$ with non-empty interior, enumerates (the infinite sequence of) all solutions of the scaled grid problem over $\mathbb{Z}[i]$ for $A$ and $\sqrt{2}^k\sqrt{5}^\ell$ in order of increasing $\ell$. Moreover, if $A$ is $M$-upright, then the algorithm requires $O(\log(1/M))$ arithmetic operations overall, plus a constant number of arithmetic operations per solution produced.*

*Proof.* This follows from Proposition 5.1.15. □

We finish this subsection with some lower bounds on the number of solutions to scaled grid problems.

*Remark* 5.1.18. If a bounded convex subset $A \subseteq \mathbb{R}^2$ contains a circle of radius $1/\sqrt{5}^k$, then the grid problem over $\mathbb{Z}[i]$ for $A$ and $\sqrt{2}^k\sqrt{5}^\ell$ has at least three solutions.

**Proposition 5.1.19.** *Let $A$ be a bounded convex subset of $\mathbb{R}^2$ with non-empty interior and assume that the scaled grid problem over $\mathbb{Z}[i]$ for $A$ and $\sqrt{2}^k\sqrt{5}^\ell$ has at least two distinct solutions. Then for all $j \geqslant 0$, the scaled grid problem over $\mathbb{Z}[i]$ for $A$ and $\sqrt{2}^k\sqrt{5}^{\ell+2j}$ has at least $5^j+1$ solutions.*

*Proof.* Let $u \neq v$ be solutions of the scaled grid problem over $\mathbb{Z}[i]$ for $A$ and $\sqrt{2}^k\sqrt{5}^\ell$. That is, $u, v \in (\sqrt{2}^k\sqrt{5}^\ell A) \cap \mathbb{Z}[i]$. For each $n = 0, 1, \ldots, 5^j$, let $\phi = \frac{n}{5^j}$, and consider $u_j = \phi u + (1-\phi)v$. Then $u_j$ is a convex combination of $u$ and $v$. Since $\sqrt{2}^k\sqrt{5}^\ell A$ is convex, it follows that $u_j \in \sqrt{2}^k\sqrt{5}^\ell A$, so that $5^j u_j$ is a solution of the scaled grid problem over $\mathbb{Z}[i]$ for $A$ and $\sqrt{2}^k\sqrt{5}^{\ell+2j}$, yielding $5^j+1$ distinct such solutions. □

## 5.2 Grid problems over $\mathbb{Z}[\omega]$

We now turn to grid problems where the lattice $L$ is a subset of $\mathbb{Z}[\omega]$. Viewing $\mathbb{C}$ as the real plane, we can consider the elements of $\mathbb{Z}[\omega]$ as points in $\mathbb{R}^2$. However, we know from Chapter 3 that $\mathbb{Z}[\omega]$ is dense in $\mathbb{C}$ and therefore does not form a lattice in the plane. To circumvent this issue, we consider subsets of $\mathbb{Z}[\omega]$ that arise as the image, under the automorphism $(-)^\bullet$, of the intersection of $\mathbb{Z}[\omega]$ and a bounded convex set $B \in \mathbb{R}^2$ with non-empty interior. We use this notion of grid to formulate grid problems over $\mathbb{Z}[\omega]$.

**Definition 5.2.1.** Let $B$ be a subset of $\mathbb{R}^2$. The *(complex) grid* for $B$ is the set

$$\operatorname{Grid}(B) = \{u \in \mathbb{Z}[\omega] \mid u^\bullet \in B\}. \tag{5.2}$$

*Remark* 5.2.2. We will only be interested in the case where $B$ is a bounded convex set with non-empty interior. In this case, the grid is discrete and infinite. It is infinite by the density of $\mathbb{Z}[\omega]$: there are infinitely many points $u \in B \cap \mathbb{Z}[\omega]$, and for each of them, $u^\bullet$ is a grid point. To see that it is discrete, recall from Remark 3.1.5 of Chapter 3 that for $u, v \in \mathbb{Z}[\omega]$ we have

$$|u - v| \cdot |u^\bullet - v^\bullet| \geqslant 1.$$

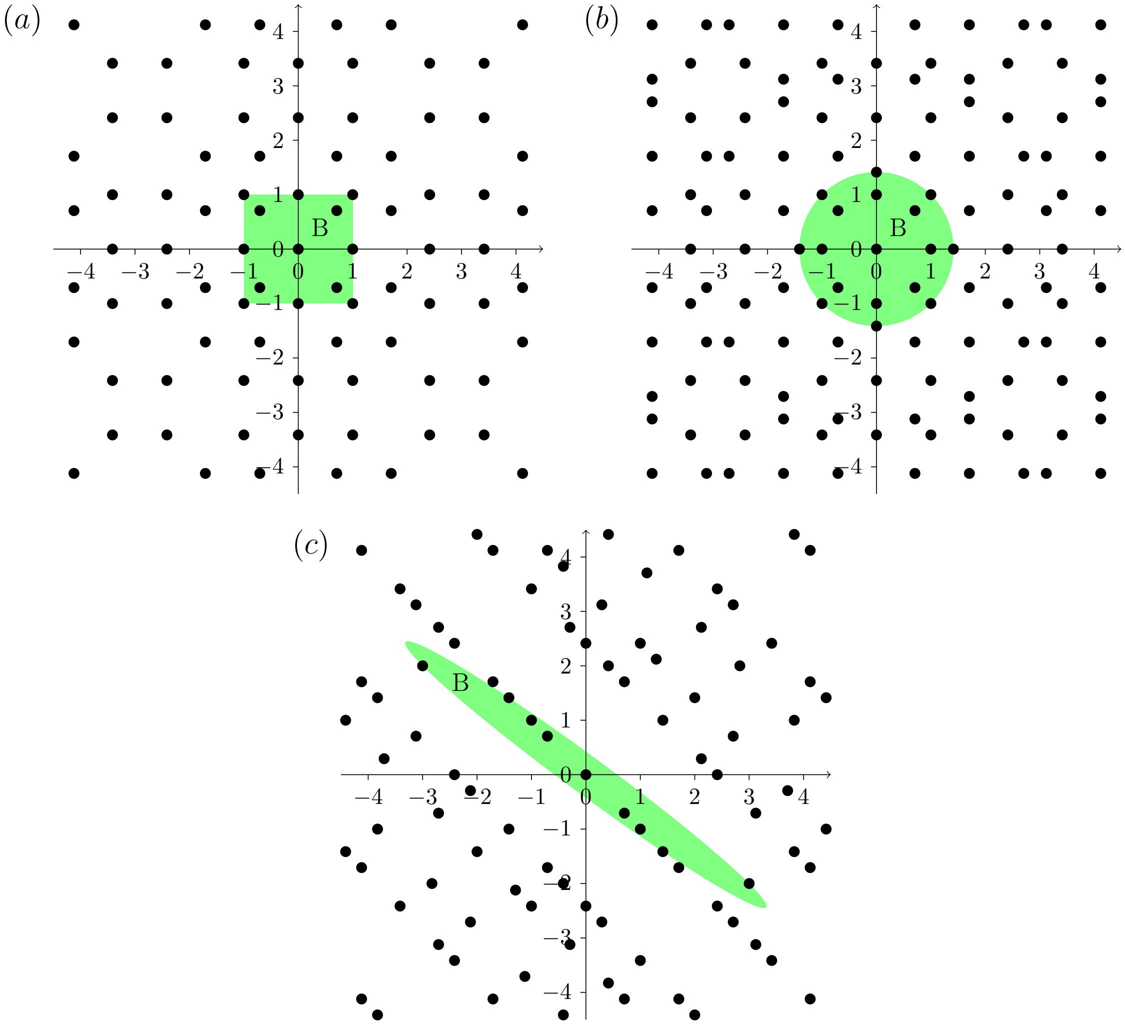


Figure 5.3: The complex grid for three different convex sets $B$. In each case, the set $B$ is shown in green and grid points are shown as black dots. (a) $B = [-1, 1]^2$. (b) $B = \{(x, y) \mid x^2 + y^2 \leqslant 2\}$. (c) $B = \{(x, y) \mid 6x^2 + 16xy + 11y^2 \leqslant 2\}$.

Since the distance between points in $B$ is bounded above, the distance between their bullets is bounded below.

Figure 5.3 illustrates the complex grids for several different convex sets $B$. Note that the grid has a 90-degree symmetry in (a), a 45-degree symmetry in (b), and a 180-degree symmetry in (c).

**Problem 5.2.3** (Grid problem over $\mathbb{Z}[\omega]$)**.** Given two bounded convex subsets $A$ and $B$ of $\mathbb{R}^2$ with non-empty interior, enumerate all the points $u \in A \cap \text{Grid}(B)$.

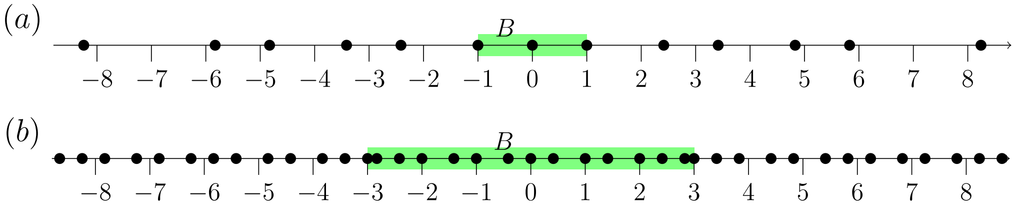


Figure 5.4: The real grid for two different intervals $B$. In both cases, the interval $B$ is shown in green, and grid points are shown as black dots.

Alternatively, grid problems over $\mathbb{Z}[\omega]$ can be understood as looking for points in $\mathbb{Z}[\omega]$ such that $u \in A$ and $u^\bullet \in B$. We also refer to the conditions $u \in A$ and $u^\bullet \in B$ as *grid constraints*. As before, an element $u \in A \cap \operatorname{Grid}(B)$ is called a *solution* to the grid problem over $\mathbb{Z}[\omega]$ for $A$ and $B$.

We solve grid problems over $\mathbb{Z}[\omega]$ by reasoning as in the previous section. That is, we first deal with the case of upright rectangles and then generalize our methods to arbitrary convex sets using ellipses.

### 5.2.1 One-dimensional grid problems

A grid problem over $\mathbb{Z}[i]$ for an upright rectangle $A$ is solved by considering the $x$ and $y$ coordinates of the problem independently. To extend this method to grid problems over $\mathbb{Z}[\omega]$, we define a one-dimensional analogue of Problem 5.2.3.

**Definition 5.2.4.** Let $B$ be a subset of $\mathbb{R}$. The *(real) grid* for $B$ is the set

$$\operatorname{Grid}(B) = \{u \in \mathbb{Z}[\sqrt{2}] \mid u^\bullet \in B\}. \tag{5.3}$$

*Remark* 5.2.5. In the following, we will only be interested in the case where $B$ is a closed interval $[y_0, y_1]$ with $y_0 < y_1$. In this case also the grid is discrete and infinite.

Figure 5.4 illustrates the grids for the intervals $[-1, 1]$ and $[-3, 3]$, respectively. For example, the first few non-negative points in $\operatorname{Grid}([-1, 1])$ are $0$, $1$, $1 + \sqrt{2}$, $2 + \sqrt{2}$, $2 + 2\sqrt{2}$, $3 + 2\sqrt{2}$, and $4 + 3\sqrt{2}$. As one would expect, the grid for $[-3, 3]$ is about three times denser than that for $[-1, 1]$. We also note that $B \subseteq B'$ implies $\operatorname{Grid}(B) \subseteq \operatorname{Grid}(B')$.

**Problem 5.2.6** (One-dimensional grid problem)**.** Given two sets of real numbers $A$ and $B$ of $\mathbb{R}$, enumerate all the points $u \in A \cap \operatorname{Grid}(B)$.

As in the two-dimensional case, Problem 5.2.6 can be equivalently expressed by the grid constraints $u \in A$ and $u^\bullet \in B$ for $u \in \mathbb{Z}[\sqrt{2}]$. In the case where $A$ and $B$ are finite intervals, the grid problem is guaranteed to have a finite number of solutions. We recall the following facts from [60].

**Lemma 5.2.7.** *Let $A = [x_0, x_1]$ and $B = [y_0, y_1]$ be closed real intervals, such that $x_1 - x_0 = \delta$ and $y_1 - y_0 = \Delta$. If $\delta\Delta < 1$, then the one-dimensional grid problem for $A$ and $B$ has at most one solution. If $\delta\Delta \geqslant (1+\sqrt{2})^2$, then the one-dimensional grid problem for $A$ and $B$ has at least one solution.*

*Proof.* Lemmas 16 and 17 of [60]. □

**Proposition 5.2.8.** *Let $A = [x_0, x_1]$ and $B = [y_0, y_1]$ be closed real intervals. There is an algorithm which enumerates all solutions to the one-dimensional grid problem for $A$ and $B$. Moreover, the algorithm only requires a constant number of arithmetic operations per solution produced.*

*Proof.* It was already noted in [60, Lemma 17] that there is an efficient algorithm for computing one solution. To see that we can efficiently enumerate all solutions, let $\delta = x_1 - x_0$ and $\Delta = y_1 - y_0$ as before. Recall from Definition 3.1.3 of Chapter 3 that $\lambda = \sqrt{2}+1$ and that $\lambda^{-1} = -\lambda^\bullet$. The grid problem for the sets $A$ and $B$ is equivalent to the grid problem for $\lambda^{-1}A$ and $-\lambda B$, because $u \in A$ and $u^\bullet \in B$ hold if and only if $\lambda^{-1}u \in \lambda^{-1}A$ and $(\lambda^{-1}u)^\bullet \in -\lambda B$. Using such rescaling, we may without loss of generality assume that $\lambda^{-1} \leqslant \delta < 1$.

Now consider any solution $u = a + b\sqrt{2} \in \mathbb{Z}[\sqrt{2}]$. From $u \in [x_0, x_1]$, we know that $x_0 - b\sqrt{2} \leqslant a \leqslant x_1 - b\sqrt{2}$. But since $x_1 - x_0 < 1$, it follows that for any $b \in \mathbb{Z}$, there is at most one $a \in \mathbb{Z}$ yielding a solution. Moreover, we note that $b = (u - u^\bullet)/\sqrt{2}^3$, so that any solution satisfies $(x_0 - y_1)/\sqrt{2}^3 \leqslant b \leqslant (x_1 - y_0)/\sqrt{2}^3$. The algorithm then proceeds by enumerating all the integers $b$ in the interval $[(x_0 - y_1)/\sqrt{2}^3, (x_1 - y_0)/\sqrt{2}^3]$. For each such $b$, find the unique integer $a$ (if any) in the interval $[x_0 - b\sqrt{2}, x_1 - b\sqrt{2}]$. Finally, check if $a + b\sqrt{2}$ is a solution. The runtime is governed by the number of $b \in \mathbb{Z}$ that need to be checked, of which there are at most $O(y_1 - y_0) = O(\delta\Delta)$. As a consequence of Lemma 5.2.7, the total number of solutions is at least $\Omega(\delta\Delta)$, and so the algorithm is efficient. □

### 5.2.2 Upright rectangles and upright sets

Recall, from Proposition 3.1.2 of Chapter 3, that $\mathbb{Z}[\omega]$ can be seen as two disjoint copies of $\mathbb{Z}[\sqrt{2}]\times\mathbb{Z}[\sqrt{2}]$. We use this fact to reduce two dimensional grid problems over $\mathbb{Z}[\omega]$ for upright rectangles $A$ and $B$ to independent one-dimensional grid problems.

**Proposition 5.2.9.** *Let $A$ and $B$ be upright rectangles. Then there is an algorithm which enumerates all the solutions to the grid problem over $\mathbb{Z}[\omega]$ for $A$ and $B$. Moreover, the algorithm requires only a constant number of arithmetic operations per solution produced.*

*Proof.* By assumption, $A = A_x \times A_y$ and $B = B_x \times B_y$, where $A_x$, $A_y$, $B_x$, and $B_y$ are closed intervals. By Proposition 3.1.2, any potential solution is of the form $u = a + bi$ or $u = a + bi + \omega$, where $a, b \in \mathbb{Z}[\sqrt{2}]$. When $u = a + bi$, then $u^\bullet = a^\bullet + b^\bullet i$. Therefore, the two-dimensional grid constraints $u \in A$ and $u^\bullet \in B$ are equivalent to the one-dimensional constraints $a \in A_x$, $a^\bullet \in B_x$ and $b \in A_y$, $b^\bullet \in B_y$. On the other hand, when $u = a + bi + \omega$, let $v = u - \omega = a + bi$. Then $v^\bullet = u^\bullet + \omega$, and the constraints $u \in A$ and $u^\bullet \in B$ are equivalent to $v \in A - \omega$ and $v^\bullet \in B + \omega$, which reduces to the one-dimensional constraints $a \in A_x - \frac{1}{\sqrt{2}}$, $a^\bullet \in B_x + \frac{1}{\sqrt{2}}$ and $b \in A_y - \frac{1}{\sqrt{2}}$, $b^\bullet \in B_y + \frac{1}{\sqrt{2}}$. In both cases, the solutions to the one-dimensional constraints can be efficiently enumerated by Proposition 5.2.8. □

Now that we are able to solve grid problem over $\mathbb{Z}[\omega]$ for upright rectangles $A$ and $B$, we can reason as in Subsection 5.1.2 to establish the following proposition.

**Proposition 5.2.10.** *Let $A, B$ be a pair of $M$-upright sets. Then there exists an algorithm which enumerates all the solutions to the grid problem over $\mathbb{Z}[\omega]$ for $A$ and $B$. Moreover, the algorithm requires $O(1/M^2)$ arithmetic operations per solution produced. In particular, when $M > 0$ is fixed, it requires only a constant number of operations per solution.*

### 5.2.3 Grid operators

We now adapt the notion of grid operator from Subsection 5.1.3 to the setting of grid problems over $\mathbb{Z}[\omega]$.

**Definition 5.2.11.** We regard $\mathbb{Z}[\omega]$ as a subset of $\mathbb{R}^2$. A real linear operator $G : \mathbb{R}^2 \to \mathbb{R}^2$ is called a *grid operator* if $G(\mathbb{Z}[\omega]) \subseteq \mathbb{Z}[\omega]$. Moreover, a grid operator $G$ is called *special* if it has determinant $\pm 1$.

Grid operators are characterized by the following lemma.

**Lemma 5.2.12.** *Let $G : \mathbb{R}^2 \to \mathbb{R}^2$ be a linear operator, which we can identify with a real $2 \times 2$-matrix. Then $G$ is a grid operator if and only if it is of the form*

$$G = \begin{bmatrix} a + \frac{a'}{\sqrt{2}} & b + \frac{b'}{\sqrt{2}} \\ c + \frac{c'}{\sqrt{2}} & d + \frac{d'}{\sqrt{2}} \end{bmatrix}, \tag{5.4}$$

*where $a, b, c, d, a', b', c', d'$ are integers satisfying $a + b + c + d \equiv 0 \pmod 2$ and $a' \equiv b' \equiv c' \equiv d' \pmod 2$.*

*Proof.* By Proposition 3.1.2 from Chapter 3, we know that a vector $u \in \mathbb{R}^2$ is in $\mathbb{Z}[\omega]$ if and only if it can be written of the form

$$u = \begin{bmatrix} x_1 + \frac{x_2}{\sqrt{2}} \\ y_1 + \frac{y_2}{\sqrt{2}} \end{bmatrix}, \tag{5.5}$$

where $x_1, x_2, y_1, y_2$ are integers and $x_2 \equiv y_2 \pmod 2$. It can then be shown by computation that every operator of the form (5.4) is a grid operator. For the converse, consider an arbitrary grid operator $G$. We prove the claim by applying $G$ to the three points $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$, $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$, and $\frac{1}{\sqrt{2}}\left[\begin{smallmatrix}1\\1\end{smallmatrix}\right] \in \mathbb{Z}[\omega]$. From $G\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right] \in \mathbb{Z}[\omega]$ and $G\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right] \in \mathbb{Z}[\omega]$, it follows that the columns of $G$ are of the form (5.5), so that $G$ is of the form (5.4), with integers $a, b, c, d, a', b', c', d'$ satisfying $a' \equiv c' \pmod 2$ and $b' \equiv d' \pmod 2$. Moreover, we have

$$G\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} \frac{a'+b'}{2} + \frac{a+b}{\sqrt{2}} \\ \frac{c'+d'}{2} + \frac{c+d}{\sqrt{2}} \end{bmatrix} \in \mathbb{Z}[\omega],$$

which implies $a + b \equiv c + d \pmod 2$ and $a' + b' \equiv c' + d' \equiv 0 \pmod 2$. Together, these conditions imply $a + b + c + d \equiv 0 \pmod 2$ and $a' \equiv b' \equiv c' \equiv d' \pmod 2$, as claimed. □

*Remark* 5.2.13. The composition of two (special) grid operators is again a (special) grid operator. If $G$ is a special grid operator, then $G$ is invertible and $G^{-1}$ is a special grid operator. If $G$ is a (special) grid operator, then $G^\bullet$ is a (special) grid operator, defined by applying $(-)^\bullet$ separately to each matrix entry, and satisfying $G^\bullet u^\bullet = (Gu)^\bullet$.

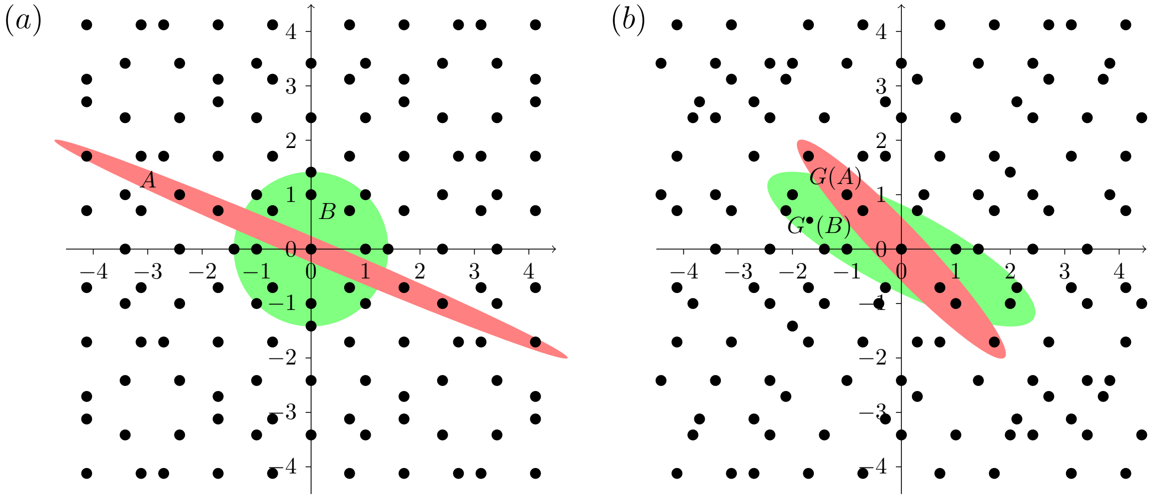


Figure 5.5: (a) The grid problem over $\mathbb{Z}[\omega]$ for two sets $A$ and $B$. (b) The grid problem over $\mathbb{Z}[\omega]$ for $G(A)$ and $G^\bullet(B)$. Note that the solutions of (a), which are the grid points in the set $A$, are in one-to-one correspondence with the solutions of (b), which are the grid points in the set $G(A)$.

---

The interest of special grid operators lies in the following fact.

**Proposition 5.2.14.** *Let $G$ be a special grid operator, and let $A$ and $B$ be subsets of $\mathbb{R}^2$. Define*

$$G(A) = \{Gu \mid u \in A\},$$
$$G^\bullet(B) = \{G^\bullet u \mid u \in B\}.$$

*Then $u \in \mathbb{Z}[\omega]$ is a solution to the grid problem over $\mathbb{Z}[\omega]$ for $A$ and $B$ if and only if $Gu$ is a solution to the grid problem over $\mathbb{Z}[\omega]$ for $G(A)$ and $G^\bullet(B)$. In particular, the grid problem over $\mathbb{Z}[\omega]$ for $A$ and $B$ is computationally equivalent to that for $G(A)$ and $G^\bullet(B)$.*

*Proof.* Let $u \in \mathbb{Z}[\omega]$. Then $u$ is a solution to the grid problem for $A$ and $B$ if and only if $u \in A$ and $u^\bullet \in B$, if and only if $Gu \in G(A)$ and $G^\bullet u^\bullet = (Gu)^\bullet \in G^\bullet(B)$, if and only if $Gu$ is a solution to the grid problem for $G(A)$ and $G^\bullet(B)$. □

Figure 5.5(a) illustrates the grid problem for a pair of sets $A$ and $B$. As before, the set $B$ is shown in green, and Grid$(B)$ is shown as black dots. The set $A$ is shown in red, and the solutions to the grid problem are the seven grid points that lie in $A$.

Figure 5.5(b) shows the grid problem for the sets $G(A)$ and $G^\bullet(B)$, where $G$ is the special grid operator

$$G = \begin{bmatrix} 1 & \sqrt{2} \\ 0 & 1 \end{bmatrix}.$$

Note that, as predicted by Proposition 5.2.14, the solutions of the transformed grid problem are in one-to-one correspondence with those of the original problem; namely, in each case, there are seven solutions.

### 5.2.4 Ellipses

Proceeding as in Subsection 5.1.4 we now prove that if $A$ and $B$ are two ellipses, then the grid problem for $A$ and $B$ over $\mathbb{Z}[\omega]$ can be solved efficiently. For this, we show that one can find a grid operator $G$ such that $G(A)$ and $G^\bullet(B)$ are *both* sufficiently upright. Indeed, we know from Proposition 5.2.10 that if both $A$ and $B$ are upright sets, then the grid problem over $\mathbb{Z}[\omega]$ for $A$ and $B$ can be solved efficiently. The fact that two ellipses have to be made simultaneously upright make the problem of finding an appropriate grid operator significantly more complicated.

We start by reformulating the problem in more convenient terms. Recall from Definition 3.1.3 of Chapter 3 that $\lambda = \sqrt{2} + 1$. The matrix $D$ corresponding to an ellipse $E$ therefore has determinant 1 if and only if it can be written in the form

$$D = \begin{bmatrix} e\lambda^{-z} & b \\ b & e\lambda^{z} \end{bmatrix} \tag{5.6}$$

for some $b, e, z \in \mathbb{R}$ with $e > 0$ and $e^2 = b^2 + 1$. As established in Proposition 5.1.10, the definition of uprightness simplifies in this case to

$$\text{up}(E) = \frac{\pi}{4e^2} = \frac{\pi}{4\sqrt{b^2+1}}. \tag{5.7}$$

Equivalently, if $\text{up}(E) = M$, then

$$b^2 = \frac{\pi^2}{16M^2} - 1. \tag{5.8}$$

Since we now have to deal with pairs of ellipses, it is convenient to introduce the following terminology for discussing pairs of matrices.

**Definition 5.2.15.** A *state* is a pair of real symmetric positive definite matrices of determinant 1. Given a state $(D, \Delta)$ with

$$D = \begin{bmatrix} e\lambda^{-z} & b \\ b & e\lambda^{z} \end{bmatrix} \qquad \Delta = \begin{bmatrix} \varepsilon\lambda^{-\zeta} & \beta \\ \beta & \varepsilon\lambda^{\zeta} \end{bmatrix} \tag{5.9}$$

we define its *skew* as $\texttt{Skew}(D, \Delta) = b^2 + \beta^2$ and its *bias* as $\texttt{Bias}(D, \Delta) = \zeta - z$.

Note that the skew of a state is small if and only if both $b^2$ and $\beta^2$ are small, which happens, by (5.7), if and only if the ellipses corresponding to $D$ and $\Delta$ both have large uprightness. So our strategy for increasing the uprightness will be to reduce the skew, as in Subsection 5.1.4. In what follows, we use $(D, \Delta)$ to denote an arbitrary state and always assume that the entries of $D$ and $\Delta$ are given as in (5.9). For future reference, we record here another useful property of states.

*Remark* 5.2.16. If $(D, \Delta)$ is a state with $b \geqslant 0$, then $-be \leqslant -b^2$. Indeed:

$$e^2 = b^2 + 1 \;\Rightarrow\; e^2 \geqslant b^2 \;\Rightarrow\; e \geqslant b \;\Rightarrow\; -be \leqslant -b^2.$$

Similarly, if $b \leqslant 0$, then $be \leqslant -b^2$. Analogous inequalities also hold for $\beta$ and $\varepsilon$.

The action of a grid operator on an ellipse can be adapted to states in a natural way, provided that the operator is special.

**Definition 5.2.17.** The action of special grid operators on states is defined as follows. Here, $G^\dagger$ denotes the transpose of $G$, and $G^\bullet$ is defined by applying $(-)^\bullet$ separately to each matrix entry, as in Remark 5.2.13.

$$(D, \Delta) \cdot G = (G^\dagger D G, G^{\bullet\dagger} \Delta G^\bullet).$$

**Lemma 5.2.18.** *Let $(D, \Delta)$ be a state, and let $A$ and $B$ be the ellipses centered at the origin that are defined by $D$ and $\Delta$, respectively. Then the ellipses $G(A)$ and $G^\bullet(B)$ are defined by the matrices $D'$ and $\Delta'$, where*

$$(D', \Delta') = (D, \Delta) \cdot G^{-1}$$

*Proof.* We have

$$\begin{aligned} G(A) &= \{G(u) \in \mathbb{R}^2 \mid u^\dagger D u \leqslant 1\} \\ &= \{v \in \mathbb{R}^2 \mid (G^{-1}v)^\dagger D (G^{-1}v) \leqslant 1\} \\ &= \{v \in \mathbb{R}^2 \mid v^\dagger (G^{-1})^\dagger D G^{-1} v \leqslant 1\}, \end{aligned}$$

so the ellipse $G(A)$ is defined by the positive operator $D' = (G^{-1})^\dagger D G^{-1}$. The proof for $G^\bullet(B)$ is similar. □

The main ingredient in our proof that states can be made upright is the following *Step Lemma.*

**Lemma 5.2.19** (Step Lemma)**.** *For any state $(D, \Delta)$, if $\texttt{Skew}(D, \Delta) \geqslant 15$, then there exists a special grid operator $G$ such that $\texttt{Skew}((D, \Delta) \cdot G) \leqslant 0.9\ \texttt{Skew}(D, \Delta)$. Moreover, $G$ can be computed using a constant number of arithmetic operations.*

Before proving the Step Lemma, we show how it can be used to derive the following proposition.

**Proposition 5.2.20.** *Let $A$ and $B$ be ellipses. Then there exists a grid operator $G$ such that $G(A)$ and $G^\bullet(B)$ are $1/6$-upright. Moreover, if $A$ and $B$ are $M$-upright, then $G$ can be efficiently computed in $O(\log(1/M))$ arithmetic operations.*

*Proof.* Let $D$ and $\Delta$ be the matrices defining $A$ and $B$ respectively, in the sense of Definition 5.1.8. Since uprightness is invariant under translations and scaling, we may without loss of generality assume that both ellipses are centered at the origin, and that $\det D = \det \Delta = 1$.

The pair $(D, \Delta)$ is a state. By applying Lemma 5.2.19 repeatedly, we get grid operators $G_1, \ldots, G_n$ such that:

$$\texttt{Skew}((D, \Delta) \cdot G_1 \ldots G_n) \leqslant 15. \tag{5.10}$$

Now let $(D', \Delta') = (D, \Delta) \cdot G_1 \ldots G_n$ and set $G = (G_1 \cdots G_n)^{-1}$. By Lemma 5.2.18, the ellipses $G(A)$ and $G^\bullet(B)$ are defined by the matrices $D'$ and $\Delta'$, respectively. Let $b$ and $\beta$ be the anti-diagonal entries of the matrices $D'$ and $\Delta'$, respectively. We have:

$$b^2 + \beta^2 = \texttt{Skew}(D', \Delta') = \texttt{Skew}((D, \Delta) \cdot G^{-1}) = \texttt{Skew}((D, \Delta) \cdot G_1 \ldots G_n) \leqslant 15,$$

hence $b^2 \leqslant 15$ and $\beta^2 \leqslant 15$. Using (5.7), we get

$$\operatorname{up}(G(A)) = \frac{\pi}{4\sqrt{b^2+1}} \geqslant \frac{\pi}{4\sqrt{16}} \geqslant 1/6$$

and similarly $\operatorname{up}(G^\bullet(B)) \geqslant 1/6$, as desired.

$$R = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \quad A = \begin{bmatrix} 1 & -2 \\ 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & \sqrt{2} \\ 0 & 1 \end{bmatrix}$$

$$K = \frac{1}{\sqrt{2}}\begin{bmatrix} -\lambda^{-1} & -1 \\ \lambda & 1 \end{bmatrix} \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Figure 5.6: The grid operators $R$, $A$, $B$, $X$, $K$, and $Z$.

To bound the number of operations, note that each application of $G_j$ reduces the skew by at least 10 percent. Therefore, the number $n$ in (5.10) satisfies $n \leqslant \log_{0.9}(15/\texttt{Skew}(D, \Delta)) = O(\log(\texttt{Skew}(D, \Delta)))$. Using (5.8), we have

$$\log(\texttt{Skew}(D, \Delta)) = \log(b^2 + \beta^2) \leqslant \log((\frac{\pi^2}{16M^2} - 1) + (\frac{\pi^2}{16M^2} - 1)) = O(\log(1/M)).$$

It follows that the computation of $G$ requires $O(\log(1/M))$ applications of the Step Lemma, each of which requires a constant number of arithmetic operations, proving the final claim of the proposition. □

The remainder of this subsection is devoted to proving the Step Lemma. To each state, we associate the pair $(z, \zeta)$. The proof of the Step Lemma is essentially a case distinction on the location of the pair $(z, \zeta)$ in the plane. We find coverings of the plane with the property that if the point $(z, \zeta)$ belongs to some region $\mathcal{O}$ of our covering, then it is easy to compute a special grid operator $G$ such that $\texttt{Skew}((D, \Delta) \cdot G) \leqslant 0.9\ \texttt{Skew}(D, \Delta)$. The relevant grid operators are given in Figure 5.6.

Each one of the next five subsections is dedicated to a particular region of the plane.

### The Shift Lemma

In this section, we consider states $(D, \Delta)$ such that $|\texttt{Bias}(D, \Delta)| > 1$. Any such state can be "shifted" to a state $(D', \Delta')$ of equal skew but with $|\texttt{Bias}(D', \Delta')| \leqslant 1$.

**Definition 5.2.21.** The *shift operators* $\sigma$ and $\tau$ are defined by:

$$\sigma = \sqrt{\lambda^{-1}}\begin{bmatrix} \lambda & 0 \\ 0 & 1 \end{bmatrix}, \tau = \sqrt{\lambda^{-1}}\begin{bmatrix} 1 & 0 \\ 0 & -\lambda \end{bmatrix}$$

Even though $\sigma$ and $\tau$ are not grid operators, we can use them to define an operation on states called a *shift by $k$*. By abuse of notation, we write this operation as an action.

**Definition 5.2.22.** Given a state $(D, \Delta)$ and $k \in \mathbb{Z}$, the *$k$-shift of* $(D, \Delta)$ is defined as:

$$(D, \Delta) \cdot \mathrm{Shift}^k = (\sigma^k D \sigma^k, \tau^k \Delta \tau^k).$$

The notation $(D, \Delta) \cdot \mathrm{Shift}^k$ is justified by the following lemma.

**Lemma 5.2.23.** *The shift of a state is a state and moreover:*

$$\texttt{Skew}((D, \Delta) \cdot \mathit{Shift}^k) = \texttt{Skew}(D, \Delta) \quad \mathit{and} \quad \texttt{Bias}((D, \Delta) \cdot \mathit{Shift}^k) = \texttt{Bias}(D, \Delta) + 2k$$

*Proof.* Compute $(D, \Delta) \cdot \mathrm{Shift}^k$:

$$\begin{aligned}
(D, \Delta) \cdot \mathrm{Shift}^k &= (\sigma^k D \sigma^k, \tau^k \Delta \tau^k) \\
&= (\sigma^k \begin{bmatrix} e\lambda^{-z} & b \\ b & e\lambda^{z} \end{bmatrix} \sigma^k, \tau^k \begin{bmatrix} \varepsilon\lambda^{-\zeta} & \beta \\ \beta & \varepsilon\lambda^{\zeta} \end{bmatrix} \tau^k) \\
&= ( \begin{bmatrix} e\lambda^{-z+k} & b \\ b & e\lambda^{z-k} \end{bmatrix}, \begin{bmatrix} \varepsilon\lambda^{-\zeta-k} & (-1)^k\beta \\ (-1)^k\beta & \varepsilon\lambda^{\zeta+k} \end{bmatrix})
\end{aligned}$$

The resulting matrices are clearly symmetric and positive definite. Moreover, since $\sigma^k$ and $\tau^k$ have determinant $\pm 1$, both $\sigma^k D \sigma^k$ and $\tau^k \Delta \tau^k$ have determinant 1. Finally:

- $\texttt{Skew}((D, \Delta) \cdot \mathrm{Shift}^k) = b^2 + ((-1)^k \beta)^2 = b^2 + \beta^2 = \texttt{Skew}(D, \Delta)$ and
- $\texttt{Bias}((D, \Delta) \cdot \mathrm{Shift}^k) = (\zeta + k) - (z - k) = \texttt{Bias}(D, \Delta) + 2k$,

which completes the proof. □

For every special grid operator $G$, there is a special grid operator $G'$ whose action on a state corresponds to shifting the state by $k$, applying $G$ and then shifting the state by $-k$.

**Lemma 5.2.24.** *If $G$ is a special grid operator and $k \in \mathbb{Z}$, then $G' = \sigma^k G \sigma^k$ is a special grid operator and moreover $G'^{\bullet} = (-\tau)^k G^{\bullet} \tau^k$.*

*Proof.* It suffices to show this for $k = 1$. Suppose $G = \begin{bmatrix} w & x \\ y & z \end{bmatrix}$ is a special grid operator and note that:

$$G' = \sigma G \sigma = \begin{bmatrix} \lambda w & x \\ y & \lambda^{-1} z \end{bmatrix} = \begin{bmatrix} \lambda^{-1} & 0 \\ 0 & \lambda^{-1} \end{bmatrix} \begin{bmatrix} \lambda & 0 \\ 0 & 1 \end{bmatrix} G \begin{bmatrix} \lambda & 0 \\ 0 & 1 \end{bmatrix}.$$

Since all the factors in the above product are grid operators, the result is also a grid operator. Moreover $\det(\sigma G \sigma) = \det(G) = 1$ so that $\sigma G \sigma$ is special. Finally:

$$G'^{\bullet} = (\sigma G \sigma)^{\bullet} = \begin{bmatrix} \lambda^{\bullet} w^{\bullet} & x^{\bullet} \\ y^{\bullet} & (\lambda^{-1})^{\bullet} z^{\bullet} \end{bmatrix} = \begin{bmatrix} -\lambda^{-1} w^{\bullet} & x^{\bullet} \\ y^{\bullet} & -\lambda z^{\bullet} \end{bmatrix} = -\tau G^{\bullet} \tau.$$

□

**Lemma 5.2.25.** *If $G$ is a grid operator, then:*

$$(((D, \Delta) \cdot \mathit{Shift}^k) \cdot G) \cdot \mathit{Shift}^k = (D, \Delta) \cdot (\sigma^k G \sigma^k).$$

*Proof.* Write $G' = \sigma^k G \sigma^k$. Simple computation then yields the result:

$$\begin{aligned}
(((D, \Delta) \cdot \mathrm{Shift}^k) \cdot G) \cdot \mathrm{Shift}^k &= ((\sigma^k D \sigma^k,\, \tau^k \Delta \tau^k) \cdot G) \cdot \mathrm{Shift}^k \\
&= (G^{\dagger} \sigma^k D \sigma^k G,\, G^{\bullet\dagger} \tau^k \Delta \tau^k G^{\bullet}) \cdot \mathrm{Shift}^k \\
&= (\sigma^k G^{\dagger} \sigma^k D \sigma^k G \sigma^k,\, \tau^k G^{\bullet\dagger} \tau^k \Delta \tau^k G^{\bullet} \tau^k) \\
&= (\sigma^k G^{\dagger} \sigma^k D \sigma^k G \sigma^k,\, ((-\tau)^k G^{\bullet\dagger} \tau^k) \Delta ((-\tau)^k G^{\bullet} \tau^k)) \\
&= (G'^{\dagger} D G',\, G'^{\bullet\dagger} \Delta G'^{\bullet}) \\
&= (D, \Delta) \cdot G' \\
&= (D, \Delta) \cdot (\sigma^k G \sigma^k).
\end{aligned}$$

□

Shifts allow us to consider only states $(D, \Delta)$ with $\texttt{Bias}(D, \Delta) \in [-1, 1]$ in the proof of the Step Lemma.

**Lemma 5.2.26.** *If the Step Lemma holds for all states $(D, \Delta)$ with $\texttt{Bias}(D, \Delta) \in [-1, 1]$, then it holds for all states.*

*Proof.* Let $(D, \Delta)$ be some state with $\texttt{Skew}(D, \Delta) \geqslant 15$. Let $x = \texttt{Bias}(D, \Delta)$ and set $k = \lfloor \frac{1-x}{2} \rfloor$. Then by Lemma 5.2.23, we have $\texttt{Skew}((D, \Delta) \cdot \mathrm{Shift}^k) = \texttt{Skew}(D, \Delta)$

and $\texttt{Bias}((D,\Delta)\cdot\mathrm{Shift}^k)\in[-1,1]$. Then by assumption, there exists a special grid operator $G$ such that $\texttt{Skew}(((D,\Delta)\cdot\mathrm{Shift}^k)\cdot G)\leqslant 0.9\ \texttt{Skew}((D,\Delta)\cdot\mathrm{Shift}^k)$. Now by Lemma 5.2.24 we know that $G'=\sigma^k\ G\ \sigma^k$ is a special grid operator. Moreover, by Lemma 5.2.25 and 5.2.23, we have:

$$\begin{array}{rcl}\texttt{Skew}((D,\Delta)\cdot G') & = & \texttt{Skew}((((D,\Delta)\cdot\mathrm{Shift}^k)\cdot G)\cdot\mathrm{Shift}^k)\\ & = & \texttt{Skew}(((D,\Delta)\cdot\mathrm{Shift}^k)\cdot G)\\ & \leqslant & 0.9\ \texttt{Skew}((D,\Delta)\cdot\mathrm{Shift}^k)\\ & = & 0.9\ \texttt{Skew}(D,\Delta),\end{array}$$

which completes the proof. □

### The $R$ Lemma

**Definition 5.2.27.** The *hyperbolic sine in base* $\lambda$ is defined as:

$$\sinh_\lambda(x)=\frac{\lambda^x-\lambda^{-x}}{2}.$$

**Lemma 5.2.28.** *Recall the operator $R$ from Figure 5.6. If $(D,\Delta)$ is a state such that $\texttt{Skew}(D,\Delta)\geqslant 15$, and such that $-0.8\leqslant z\leqslant 0.8$ and $-0.8\leqslant\zeta\leqslant 0.8$, then:*

$$\texttt{Skew}((D,\Delta)\cdot R)\leqslant 0.9\ \texttt{Skew}(D,\Delta).$$

*Proof.* Compute the action of $R$ on $(D,\Delta)$:

$$\begin{array}{rcl}R^\dagger DR & = & \frac{1}{2}\begin{bmatrix}1 & 1\\ -1 & 1\end{bmatrix}\begin{bmatrix}e\lambda^{-z} & b\\ b & e\lambda^{z}\end{bmatrix}\begin{bmatrix}1 & -1\\ 1 & 1\end{bmatrix}\\ & = & \begin{bmatrix}\cdots & \frac{e(\lambda^z-\lambda^{-z})}{2}\\ \frac{e(\lambda^z-\lambda^{-z})}{2} & \cdots\end{bmatrix}=\begin{bmatrix}\cdots & e\sinh_\lambda(z)\\ e\sinh_\lambda(z) & \cdots\end{bmatrix},\\ R^{\bullet\dagger}\Delta R^{\bullet} & = & \frac{1}{2}\begin{bmatrix}-1 & -1\\ 1 & -1\end{bmatrix}\begin{bmatrix}\varepsilon\lambda^{-\zeta} & \beta\\ \beta & \varepsilon\lambda^{\zeta}\end{bmatrix}\begin{bmatrix}-1 & 1\\ -1 & -1\end{bmatrix}\\ & = & \begin{bmatrix}\cdots & \frac{\varepsilon(\lambda^\zeta-\lambda^{-\zeta})}{2}\\ \frac{\varepsilon(\lambda^\zeta-\lambda^{-\zeta})}{2} & \cdots\end{bmatrix}=\begin{bmatrix}\cdots & \varepsilon\sinh_\lambda(\zeta)\\ \varepsilon\sinh_\lambda(\zeta) & \cdots\end{bmatrix}.\end{array}$$

Therefore $\texttt{Skew}((D,\Delta)\cdot R)=e^2\sinh^2_\lambda(z)+\varepsilon^2\sinh^2_\lambda(\zeta)$. But recall that $e^2=b^2+1$ and $\varepsilon^2=\beta^2+1$, so that in fact:

$$\texttt{Skew}((D,\Delta)\cdot R)=(b^2+1)\sinh^2_\lambda(z)+(\beta^2+1)\sinh^2_\lambda(\zeta).$$

We assumed $-0.8 \leqslant z, \zeta \leqslant 0.8$ and this implies that $\sinh_\lambda^2(\zeta), \sinh_\lambda^2(z) \leqslant \sinh_\lambda^2(0.8)$. Writing $y = \sinh_\lambda^2(0.8)$ for brevity, and using the assumption that $\texttt{Skew}(D,\Delta) \geqslant 15$, we get:

$$\begin{aligned}
\texttt{Skew}((D,\Delta)\cdot R) &= (b^2+1)\sinh_\lambda^2(z) + (\beta^2+1)\sinh_\lambda^2(\zeta) \\
&\leqslant (b^2+1)y + (\beta^2+1)y \\
&= (b^2+\beta^2+2)y \\
&\leqslant \texttt{Skew}(D,\Delta)(1+\tfrac{2}{15})y.
\end{aligned}$$

This completes the proof, since $(1+\frac{2}{15})y = (1+\frac{2}{15})\sinh_\lambda^2(0.8) \approx 0.663 \leqslant 0.9$. □

### The $K$ Lemma

**Definition 5.2.29.** The *hyperbolic cosine in base* $\lambda$ is defined as:

$$\cosh_\lambda(x) = \frac{\lambda^x + \lambda^{-x}}{2}.$$

**Lemma 5.2.30.** *Recall the operator $K$ from Figure 5.6. If $(D,\Delta)$ is a state such that $\texttt{Bias}(D,\Delta) \in [-1,1]$, $\texttt{Skew}(D,\Delta) \geqslant 15$, and such that $b, \beta \geqslant 0$, $z \leqslant 0.3$, and $0.8 \leqslant \zeta$, then:*

$$\texttt{Skew}((D,\Delta)\cdot K) \leqslant 0.9\ \texttt{Skew}(D,\Delta).$$

*Proof.* Compute the action of $K$ on $(D,\Delta)$:

$$\begin{aligned}
&K^\dagger D K \\
&\quad= \frac{1}{2}\begin{bmatrix} -\lambda^{-1} & \lambda \\ -1 & 1 \end{bmatrix}\begin{bmatrix} e\lambda^{-z} & b \\ b & e\lambda^{z} \end{bmatrix}\begin{bmatrix} -\lambda^{-1} & -1 \\ \lambda & 1 \end{bmatrix} \\
&\quad= \frac{1}{2}\begin{bmatrix} \cdots & e(\lambda^{z+1}+\lambda^{-z-1}) - 2\sqrt{2}b \\ e(\lambda^{z+1}+\lambda^{-z-1}) - 2\sqrt{2}b & \cdots \end{bmatrix} \\
&\quad= \begin{bmatrix} \cdots & e\cosh_\lambda(z+1) - \sqrt{2}b \\ e\cosh_\lambda(z+1) - \sqrt{2}b & \cdots \end{bmatrix},
\end{aligned}$$

$$
\begin{aligned}
& K^{\bullet\dagger}\Delta K^{\bullet} \\
= \; & \frac{1}{2}\begin{bmatrix} \lambda & -\lambda^{-1} \\ -1 & 1 \end{bmatrix}\begin{bmatrix} \varepsilon\lambda^{-\zeta} & \beta \\ \beta & \varepsilon\lambda^{\zeta} \end{bmatrix}\begin{bmatrix} \lambda & -1 \\ -\lambda^{-1} & 1 \end{bmatrix} \\
= \; & \frac{1}{2}\begin{bmatrix} \dots & -\varepsilon(\lambda^{\zeta-1}+\lambda^{-\zeta+1})+2\sqrt{2}\beta \\ -\varepsilon(\lambda^{\zeta-1}+\lambda^{-\zeta+1})+2\sqrt{2}\beta & \dots \end{bmatrix} \\
= \; & \begin{bmatrix} \dots & \sqrt{2}\beta-\varepsilon\cosh_{\lambda}(\zeta-1) \\ \sqrt{2}\beta-\varepsilon\cosh_{\lambda}(\zeta-1) & \dots \end{bmatrix}.
\end{aligned}
$$

Therefore:

$$
\mathtt{Skew}((D,\Delta)\cdot K) = (\sqrt{2}b - e\cosh_{\lambda}(z+1))^2 + (\sqrt{2}\beta - \varepsilon\cosh_{\lambda}(\zeta-1))^2. \qquad (5.11)
$$

But recall that $e^2 = b^2+1$, and from Remark 5.2.16 that $b \geqslant 0$ implies $-be \leqslant -b^2$, so:

$$
\begin{aligned}
& (\sqrt{2}b - e\cosh_{\lambda}(z+1))^2 \\
= \; & 2b^2 - 2\sqrt{2}\,be\cosh_{\lambda}(z+1) + e^2\cosh^2_{\lambda}(z+1) \\
\leqslant \; & 2b^2 - 2\sqrt{2}\,b^2\cosh_{\lambda}(z+1) + (b^2+1)\cosh^2_{\lambda}(z+1) \\
= \; & b^2(2 - 2\sqrt{2}\cosh_{\lambda}(z+1) + \cosh^2_{\lambda}(z+1)) + \cosh^2_{\lambda}(z+1) \\
= \; & b^2(\sqrt{2} - \cosh_{\lambda}(z+1))^2 + \cosh^2_{\lambda}(z+1). \qquad (5.12)
\end{aligned}
$$

Reasoning analogously, we also have

$$
(\sqrt{2}\beta - \varepsilon\cosh_{\lambda}(\zeta-1))^2 \leqslant \beta^2(\sqrt{2} - \cosh_{\lambda}(\zeta-1))^2 + \cosh^2_{\lambda}(\zeta-1). \qquad (5.13)
$$

By assumption, $\mathtt{Bias}(D,\Delta) \in [-1,1]$, thus $\zeta \leqslant z+1$. This, together with the assumptions $0.8 \leqslant \zeta$ and $z \leqslant 0.3$, implies that both $z+1$ and $\zeta-1$ are in the interval $[-0.2, 1.3]$. On this interval, the function $\cosh^2_{\lambda}(x)$ assumes its maximum at $x = 1.3$, and the function $f(x) = (\sqrt{2} - \cosh_{\lambda}(x))^2$ assumes its maximum at $x = 0$. Therefore,

$$
b^2(\sqrt{2} - \cosh_{\lambda}(z+1))^2 + \cosh^2_{\lambda}(z+1) \leqslant b^2(\sqrt{2} - \cosh_{\lambda}(0))^2 + \cosh^2_{\lambda}(1.3) \qquad (5.14)
$$

and

$$
\beta^2(\sqrt{2} - \cosh_{\lambda}(\zeta-1))^2 + \cosh^2_{\lambda}(\zeta-1) \leqslant \beta^2(\sqrt{2} - \cosh_{\lambda}(0))^2 + \cosh^2_{\lambda}(1.3). \qquad (5.15)
$$

Combining (5.11)–(5.15), together with the assumption that $\texttt{Skew}(D, \Delta) \geqslant 15$, yields:

$$\begin{aligned}
\texttt{Skew}((D,\Delta)\cdot K) &= (\sqrt{2}b - e\cosh_\lambda(z+1))^2 + (\sqrt{2}\beta - \varepsilon\cosh_\lambda(\zeta-1))^2 \\
&\leqslant (b^2+\beta^2)(\sqrt{2}-\cosh_\lambda(0))^2 + 2\cosh^2_\lambda(1.3) \\
&= \texttt{Skew}(D,\Delta)(\sqrt{2}-\cosh_\lambda(0))^2 + 2\cosh^2_\lambda(1.3) \\
&\leqslant \texttt{Skew}(D,\Delta)((\sqrt{2}-\cosh_\lambda(0))^2 + \frac{2}{15}\cosh^2_\lambda(1.3))
\end{aligned}$$

This completes the proof since $(\sqrt{2}-\cosh_\lambda(0))^2 + \frac{2}{15}\cosh^2_\lambda(1.3) \approx 0.571 \leqslant 0.9$. □

### The $A$ Lemma

**Definition 5.2.31.** Let $g(x) = (1-2x)^2$.

**Lemma 5.2.32.** *Recall the operator $A$ from Figure 5.6. If $(D,\Delta)$ is a state such that $\texttt{Bias}(D,\Delta) \in [-1,1]$, $\texttt{Skew}(D,\Delta) \geqslant 15$, and such that $b, \beta \geqslant 0$ and $0.3 \leqslant z, \zeta$, then there exists $n \in \mathbb{Z}$ such that:*

$$\texttt{Skew}((D,\Delta)\cdot A^n) \leqslant 0.9\ \texttt{Skew}(D,\Delta).$$

*Proof.* Let $c = \min\{z,\zeta\}$ and $n = \max\{1, \lfloor\frac{\lambda^c}{2}\rfloor\}$. Compute the action of $A^n$ on $(D,\Delta)$:

$$\begin{aligned}
A^{n\dagger}DA^n &= \begin{bmatrix} 1 & 0 \\ -2n & 1 \end{bmatrix}\begin{bmatrix} e\lambda^{-z} & b \\ b & e\lambda^{z} \end{bmatrix}\begin{bmatrix} 1 & -2n \\ 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \dots & b-2ne\lambda^{-z} \\ b-2ne\lambda^{-z} & \dots \end{bmatrix}, \\
A^{n\bullet\dagger}\Delta A^{n\bullet} &= A^{n\dagger}\Delta A^n \\
&= \begin{bmatrix} \dots & \beta-2n\varepsilon\lambda^{-\zeta} \\ \beta-2n\varepsilon\lambda^{-\zeta} & \dots \end{bmatrix}.
\end{aligned}$$

Therefore:

$$\texttt{Skew}((D,\Delta)\cdot A^n) = (b-2ne\lambda^{-z})^2 + (\beta-2n\varepsilon\lambda^{-\zeta})^2$$

But recall that $e^2 = b^2+1$ and $\varepsilon^2 = \beta^2+1$, and from Remark 5.2.16 that $b,\beta \geqslant 0$ implies $-be \leqslant -b^2$ and $-\varepsilon\beta \leqslant -\beta^2$. Using these facts, we can expand the above formula as follows:

$$\begin{aligned}
&\texttt{Skew}((D,\Delta)\cdot A^n)\\
&\quad= (b-2ne\lambda^{-z})^2+(\beta-2n\varepsilon\lambda^{-\zeta})^2\\
&\quad= b^2-4nbe\lambda^{-z}+4n^2e^2\lambda^{-2z}+\beta^2-4n\beta\varepsilon\lambda^{-\zeta}+4n^2\varepsilon^2\lambda^{-2\zeta}\\
&\quad\leqslant b^2-4nb^2\lambda^{-z}+4n^2(b^2+1)\lambda^{-2z}+\beta^2-4n\beta^2\lambda^{-\zeta}+4n^2(\beta^2+1)\lambda^{-2\zeta}\\
&\quad= b^2(1-4n\lambda^{-z}+4n^2\lambda^{-2z})+\beta^2(1-4n\lambda^{-\zeta}+4n^2\lambda^{-2\zeta})+4n^2(\lambda^{-2z}+\lambda^{-2\zeta})\\
&\quad= b^2(1-2n\lambda^{-z})^2+\beta^2(1-2n\lambda^{-\zeta})^2+4n^2(\lambda^{-2z}+\lambda^{-2\zeta})\\
&\quad= b^2g(n\lambda^{-z})+\beta^2g(n\lambda^{-\zeta})+4n^2(\lambda^{-2z}+\lambda^{-2\zeta}).
\end{aligned}$$

Writing $y=\max\{g(n\lambda^{-z}),g(n\lambda^{-\zeta})\}$ for brevity, and using the assumption that $\texttt{Skew}(D,\Delta)\geqslant 15$ together with the fact that $c\leqslant z,\zeta$, we get:

$$\begin{aligned}
\texttt{Skew}((D,\Delta)\cdot A^n) &\leqslant b^2y+\beta^2y+8n^2\lambda^{-2c}\\
&= \texttt{Skew}(D,\Delta)y+8n^2\lambda^{-2c}\\
&\leqslant \texttt{Skew}(D,\Delta)(y+\frac{8}{15}n^2\lambda^{-2c}).
\end{aligned}$$

To finish the proof, it remains to show that $y+\frac{8}{15}n^2\lambda^{-2c}\leqslant 0.9$. There are two cases:

- If $\lfloor\frac{\lambda^c}{2}\rfloor\geqslant 1$, then $\frac{\lambda^c}{4}\leqslant n\leqslant\frac{\lambda^c}{2}$. From $n\leqslant\frac{\lambda^c}{2}$, we have $2n\lambda^{-c}\leqslant 1$, and so $\frac{8}{15}n^2\lambda^{-2c}\leqslant\frac{2}{15}$. Moreover, because $\texttt{Bias}(D,\Delta)\in[-1,1]$, we have $c\leqslant z,\zeta\leqslant c+1$. Hence $\frac{1}{4\lambda}=\frac{\lambda^c}{4}\lambda^{-c-1}\leqslant n\lambda^{-c-1}\leqslant n\lambda^{-z},n\lambda^{-\zeta}\leqslant n\lambda^{-c}\leqslant\frac{1}{2}$. On the interval $[\frac{1}{4\lambda},\frac{1}{2}]$, the function $g(x)$ assumes its maximum at $x=\frac{1}{4\lambda}$. This implies that $y\leqslant g(\frac{1}{4\lambda})$. This completes the present case since we get:
$$y+\frac{8}{15}n^2\lambda^{-2c}\leqslant g(\frac{1}{4\lambda})+\frac{2}{15}\approx 0.762\leqslant 0.9.$$
- If $\lfloor\frac{\lambda^c}{2}\rfloor<1$, then $n=1$ and $\lambda^c<2$. From $0.3\leqslant c$, we have $\frac{8}{15}n^2\lambda^{-2c}\leqslant\frac{8}{15}\lambda^{-0.6}$. Moreover, because $\texttt{Bias}(D,\Delta)\in[-1,1]$, we have $0.3\leqslant c\leqslant z,\zeta\leqslant c+1$. With $\lambda^c\leqslant 2$, this implies that $\frac{1}{2\lambda}\leqslant\lambda^{-c-1}\leqslant\lambda^{-z},\lambda^{-\zeta}\leqslant\lambda^{-0.3}$. Therefore both $\lambda^{-z}$ and $\lambda^{-\zeta}$ are in the interval $[\frac{1}{2\lambda},\lambda^{-0.3}]$. On this interval, the function $g(x)$ assumes its maximum at $x=\frac{1}{2\lambda}$, and therefore $y\leqslant g(\frac{1}{2\lambda})$. This completes the proof since:
$$y+\frac{8}{15}n^2\lambda^{-2c}\leqslant g(\frac{1}{2\lambda})+\frac{8}{15}\lambda^{-0.6}\approx 0.657\leqslant 0.9.$$

□

### The $B$ Lemma

**Definition 5.2.33.** Let $h(x) = (1-\sqrt{2}x)^2$.

**Lemma 5.2.34.** *Recall the operator $B$ from Figure 5.6. If $(D,\Delta)$ is a state such that $\texttt{Bias}(D,\Delta) \in [-1,1]$, $\texttt{Skew}(D,\Delta) \geqslant 15$, and such that $b \leqslant 0 \leqslant \beta$ and $-0.2 \leqslant z, \zeta$, then there exists $n \in \mathbb{Z}$ such that:*

$$\texttt{Skew}((D,\Delta)\cdot B^n) \leqslant 0.9\ \texttt{Skew}(D,\Delta).$$

*Proof.* Let $c = \min\{z,\zeta\}$, $n = \max\{1, \lfloor \frac{\lambda^c}{\sqrt{2}} \rfloor\}$ and compute the action of $B^n$ on $(D,\Delta)$:

$$\begin{aligned}
B^{n\dagger} D B^n &= \begin{bmatrix} 1 & 0 \\ \sqrt{2}n & 1 \end{bmatrix} \begin{bmatrix} e\lambda^{-z} & b \\ b & e\lambda^{z} \end{bmatrix} \begin{bmatrix} 1 & \sqrt{2}n \\ 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \dots & b+\sqrt{2}\,ne\lambda^{-z} \\ b+\sqrt{2}\,ne\lambda^{-z} & \dots \end{bmatrix}, \\
B^{n\bullet\dagger} D B^{n\bullet} &= \begin{bmatrix} 1 & 0 \\ -\sqrt{2}n & 1 \end{bmatrix} \begin{bmatrix} \varepsilon\lambda^{-\zeta} & \beta \\ \beta & \varepsilon\lambda^{\zeta} \end{bmatrix} \begin{bmatrix} 1 & -\sqrt{2}n \\ 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \dots & \beta-\sqrt{2}\,n\varepsilon\lambda^{-\zeta} \\ \beta-\sqrt{2}\,n\varepsilon\lambda^{-\zeta} & \dots \end{bmatrix}.
\end{aligned}$$

Therefore:

$$\texttt{Skew}((D,\Delta)\cdot B^n) = (b+\sqrt{2}\,ne\lambda^{-z})^2 + (\beta-\sqrt{2}\,n\varepsilon\lambda^{-\zeta})^2.$$

But recall that $e^2 = b^2+1$, that $\varepsilon^2 = \beta^2+1$, and from Remark 5.2.16 that $b \leqslant 0 \leqslant \beta$ implies $be \leqslant -b^2$ and $-\beta\varepsilon \leqslant -\beta^2$. Using these facts, we can expand the above formula as follows:

$$\begin{aligned}
&\texttt{Skew}((D,\Delta)\cdot B^n) \\
&\quad= (b+\sqrt{2}\,ne\lambda^{-z})^2 + (\beta-\sqrt{2}\,n\varepsilon\lambda^{-\zeta})^2 \\
&\quad= b^2 + 2\sqrt{2}\,nbe\lambda^{-z} + 2n^2e^2\lambda^{-2z} + \beta^2 - 2\sqrt{2}\,n\beta\varepsilon\lambda^{-\zeta} + 2n^2\varepsilon^2\lambda^{-2\zeta} \\
&\quad\leqslant b^2 - 2\sqrt{2}\,nb^2\lambda^{-z} + 2n^2(b^2+1)\lambda^{-2z} + \beta^2 - 2\sqrt{2}\,n\beta^2\lambda^{-\zeta} + 2n^2(\beta^2+1)\lambda^{-2\zeta} \\
&\quad= b^2(1-2\sqrt{2}\,n\lambda^{-z}+2n^2\lambda^{-2z}) + \beta^2(1-2\sqrt{2}\,n\lambda^{-\zeta}+2n^2\lambda^{-2\zeta}) + 2n^2(\lambda^{-2z}+\lambda^{-2\zeta}) \\
&\quad= b^2(1-\sqrt{2}\,n\lambda^{-z})^2 + \beta^2(1-\sqrt{2}\,n\lambda^{-\zeta})^2 + 2n^2(\lambda^{-2z}+\lambda^{-2\zeta}). \\
&\quad= b^2h(n\lambda^{-z}) + \beta^2h(n\lambda^{-\zeta}) + 2n^2(\lambda^{-2z}+\lambda^{-2\zeta}).
\end{aligned}$$

Writing $y = \max\{h(n\lambda^{-z}), h(n\lambda^{-\zeta})\}$ for brevity, and using the assumption that $\mathtt{Skew}(D,\Delta) \geqslant 15$, together with the fact that $c \leqslant z, \zeta$, we get:

$$\begin{aligned}\mathtt{Skew}((D,\Delta)\cdot B^n) &\leqslant b^2y + \beta^2y + 4n^2\lambda^{-2c}\\ &= \mathtt{Skew}(D,\Delta)y + 4n^2\lambda^{-2c}\\ &\leqslant \mathtt{Skew}(D,\Delta)(y + \frac{4}{15}n^2\lambda^{-2c}).\end{aligned}$$

To finish the proof, it remains to show that $y + \frac{4}{15}n^2\lambda^{-2c} \leqslant 0.9$. There are two cases:

- If $\lfloor \frac{\lambda^c}{\sqrt{2}} \rfloor \geqslant 1$, then $\frac{\lambda^c}{2\sqrt{2}} \leqslant n \leqslant \frac{\lambda^c}{\sqrt{2}}$. From $n \leqslant \frac{\lambda^c}{\sqrt{2}}$, we have $2n^2\lambda^{-2c} \leqslant 1$, and so $\frac{4n^2\lambda^{-2c}}{15} \leqslant \frac{2}{15}$. Moreover, because $\mathtt{Bias}(D,\Delta) \in [-1,1]$, we have $c \leqslant z, \zeta \leqslant c+1$. Hence $\frac{1}{2\sqrt{2}\,\lambda} = \frac{\lambda^c}{2\sqrt{2}}\lambda^{-c-1} \leqslant n\lambda^{-c-1} \leqslant n\lambda^{-z}, n\lambda^{-\zeta} \leqslant n\lambda^{-c} \leqslant \frac{1}{\sqrt{2}}$. On the interval $[\frac{1}{2\sqrt{2}\,\lambda}, \frac{1}{\sqrt{2}}]$, the function $h(x)$ assumes its maximum at $x = \frac{1}{2\sqrt{2}\,\lambda}$. This implies that $y \leqslant h(\frac{1}{2\sqrt{2}\,\lambda})$. This completes the present case since we get:

  $$y + \frac{4}{15}n^2\lambda^{-2c} \leqslant h(\frac{1}{2\sqrt{2}\,\lambda}) + \frac{2}{15} \approx 0.762 \leqslant 0.9.$$

- If $\lfloor \frac{\lambda^c}{\sqrt{2}} \rfloor < 1$, then $n = 1$ and $\lambda^c < \sqrt{2}$. From $-0.2 \leqslant c$, we have $\frac{4}{15}n^2\lambda^{-2c} \leqslant \frac{4}{15}\lambda^{0.4}$. Moreover, because $\mathtt{Bias}(D,\Delta) \in [-1,1]$, we have $-0.2 \leqslant c \leqslant z, \zeta \leqslant c+1$. With $\lambda^c \leqslant \sqrt{2}$, this implies that $\frac{1}{\sqrt{2}\,\lambda} \leqslant \lambda^{-c-1} \leqslant \lambda^{-z}, \lambda^{-\zeta} \leqslant \lambda^{0.2}$. Therefore both $\lambda^{-z}$ and $\lambda^{-\zeta}$ are in the interval $[\frac{1}{\sqrt{2}\,\lambda}, \lambda^{0.2}]$. On this interval, the function $h(x)$ assumes its maximum at $x = \lambda^{0.2}$, and therefore $y \leqslant h(\lambda^{0.2})$. This completes the proof since:

  $$y + \frac{4}{15}n^2\lambda^{-2c} \leqslant h(\lambda^{0.2}) + \frac{4}{15}\lambda^{0.4} \approx 0.851 \leqslant 0.9.$$

□

### Proof of the Step Lemma

The proof of the Step Lemma is now basically a case distinction, using the cases enumerated in lemmas 5.2.26–5.2.34, as well as some additional symmetric cases. In particular, the following remark will allow us to use the grid operators $X$ and $Z$ to reduce the number of cases to consider.

*Remark* 5.2.35. The grid operator $Z$ negates the anti-diagonal entries of a state while the operator $X$ swaps the diagonal entries of a state. This follows by simple computation since

$$(D,\Delta)\cdot Z = \left(\begin{bmatrix} e\lambda^{-z} & -b \\ -b & e\lambda^{z} \end{bmatrix}, \begin{bmatrix} \varepsilon\lambda^{-\zeta} & -\beta \\ -\beta & \varepsilon\lambda^{\zeta} \end{bmatrix}\right)$$

and

$$(D,\Delta)\cdot X = \left(\begin{bmatrix} e\lambda^{z} & b \\ b & e\lambda^{-z} \end{bmatrix}, \begin{bmatrix} \varepsilon\lambda^{\zeta} & \beta \\ \beta & \varepsilon\lambda^{-\zeta} \end{bmatrix}\right).$$

Moreover, $\texttt{Bias}((D,\Delta)\cdot Z) = \texttt{Bias}(D,\Delta)$ and $\texttt{Bias}((D,\Delta)\cdot X) = -\texttt{Bias}(D,\Delta)$.

**Lemma** (Step Lemma)**.** *For any state* $(D,\Delta)$*, if* $\texttt{Skew}(D,\Delta) \geqslant 15$*, then there exists a special grid operator* $G$ *such that* $\texttt{Skew}((D,\Delta)\cdot G) \leqslant 0.9\ \texttt{Skew}(D,\Delta)$*. Moreover,* $G$ *can be computed using a constant number of arithmetic operations.*

*Proof.* Let $(D,\Delta)$ be a state such that $\texttt{Skew}(D,\Delta) \geqslant 15$. By Lemma 5.2.26 we can assume w.l.o.g. that $\texttt{Bias}(D,\Delta) \in [-1,1]$. Moreover, by Remark 5.2.35, we can also assume that $\beta \geqslant 0$ and $z+\zeta \geqslant 0$. Note that the application of the grid operators $X$ and/or $Z$ in Remark 5.2.35 preserves the fact that $\texttt{Bias}(D,\Delta) \in [-1,1]$. We now treat in turn the cases $b \geqslant 0$ and $b \leqslant 0$.

**Case 1** $b \geqslant 0$. A covering of the strip defined by $z-\zeta \in [-1,1]$ and $z+\zeta \geqslant 0$ is depicted in Figure 5.7(a). The $R$ region (in green) and the $A$ region (in red) are defined as the intersection of this space with $\{(z,\zeta) \mid -0.8 \leqslant z,\zeta \leqslant 0.8\}$ and $\{(z,\zeta) \mid z \leqslant 0.3 \text{ and } 0.8 \leqslant \zeta\}$ respectively. The $K$ and $K^\bullet$ regions (both in blue) fill the remaining space.

We now consider in turn the possible locations of the pair $(z,\zeta)$ in this covering.

1. If $-0.8 \leqslant z,\zeta \leqslant 0.8$, then by Lemma 5.2.28 we have $\texttt{Skew}((D,\Delta)\cdot R) \leqslant 0.9\ \texttt{Skew}(D,\Delta)$.
2. If $z \leqslant 0.3$ and $0.8 \leqslant \zeta$, then by Lemma 5.2.30 we have $\texttt{Skew}((D,\Delta)\cdot K) \leqslant 0.9\ \texttt{Skew}(D,\Delta)$.
3. If $0.3 \leqslant z,\zeta$, then there exists $n \in \mathbb{Z}$ such that $\texttt{Skew}((D,\Delta)\cdot A^n) \leqslant 0.9\ \texttt{Skew}(D,\Delta)$ by Lemma 5.2.32.

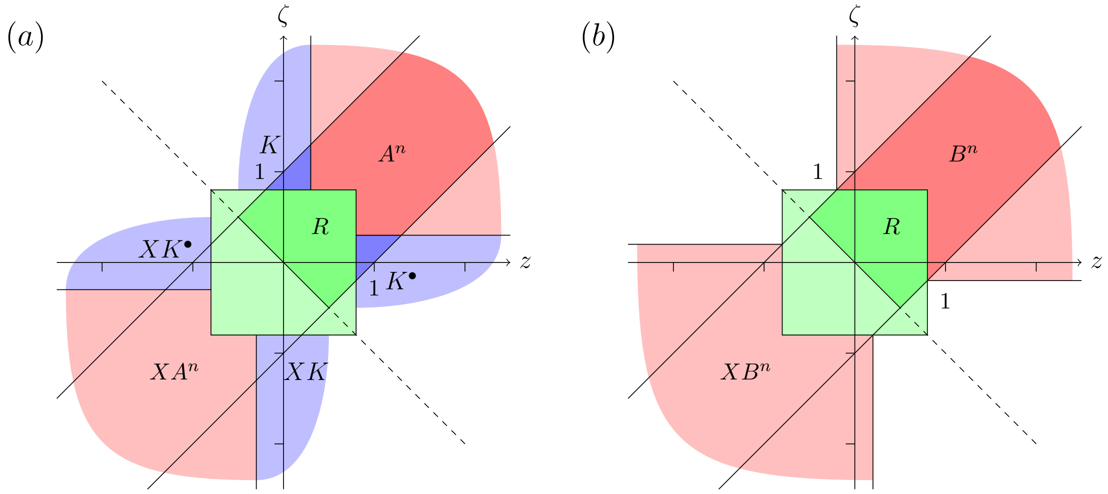


Figure 5.7: (a) A covering of the region $z - \zeta \in [-1, 1]$ and $z + \zeta \geqslant 0$ for the case $b \geqslant 0$. (b) A covering of the region $z - \zeta \in [-1, 1]$ and $z + \zeta \geqslant 0$ for the case $b \leqslant 0$.

4. If $0.8 \leqslant z$ and $\zeta \leqslant 0.3$, then note that $(D, \Delta) \cdot K^\bullet = (\Delta, D) \cdot K$, and therefore $\texttt{Skew}((D, \Delta) \cdot K) \leqslant 0.9\, \texttt{Skew}(D, \Delta)$ by Lemma 5.2.30:

$$\texttt{Skew}((D, \Delta) \cdot K^\bullet) = \texttt{Skew}((\Delta, D) \cdot K) \leqslant 0.9\, \texttt{Skew}(\Delta, D) = 0.9\, \texttt{Skew}(D, \Delta).$$

**Case 2** $b \leqslant 0$. As above, we use a covering of the strip defined by $z - \zeta \in [-1, 1]$ and $z + \zeta \geqslant 0$ and consider the possible locations of $(z, \zeta)$ in this space. The relevant covering is depicted in Figure 5.7(b), where the $R$ region (in green) is defined as above and the $B$ region (in red) is defined as the intersection of the strip with $\{(z, \zeta) \mid z, \zeta \geqslant -0.2\}$.

1. If $-0.8 \leqslant z, \zeta \leqslant 0.8$, then by Lemma 5.2.28 we have $\texttt{Skew}((D, \Delta) \cdot R) \leqslant 0.9\, \texttt{Skew}(D, \Delta)$.

2. If $z, \zeta \geqslant -0.2$ then there exists $n \in \mathbb{Z}$ such that $\texttt{Skew}((D, \Delta) \cdot B^n) \leqslant 0.9\, \texttt{Skew}(D, \Delta)$ by Lemma 5.2.34.

Finally, note that only a constant number of calculations are required to decide which of the above cases applies. Moreover, each case only requires a constant number of operations. Specifically, the computation of $k$ and $\sigma^k$ in Lemma 5.2.26, of $n$ and $A^n$

in Lemma 5.2.32, and of $n$ and $B^n$ in Lemma 5.2.34 each require just a fixed number of operations, and each of the remaining cases produces a fixed grid operator. □

### 5.2.5 General solution to grid problems over $\mathbb{Z}[\omega]$

We are finally in a position to solve Problem 5.2.3.

**Proposition 5.2.36.** *There is an algorithm which, given two bounded convex subset $A$ and $B$ of $\mathbb{R}^2$ with non-empty interior, enumerates all solutions of the grid problem over $\mathbb{Z}[\omega]$ for $A$ and $B$. Moreover, if $A$ and $B$ are $M$-upright, then the algorithm requires $O(\log(1/M))$ arithmetic operations overall, plus a constant number of arithmetic operations per solution produced.*

*Proof.* Analogous to the proof of Proposition 5.1.15, using Proposition 5.2.20 to find the appropriate grid operator. □

### 5.2.6 Scaled grid problems over $\mathbb{Z}[\omega]$

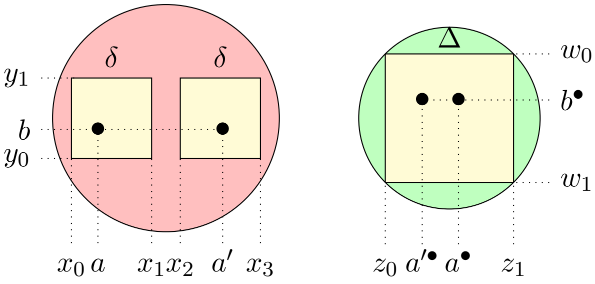
We close this chapter by considering scaled versions of Problem 5.2.3, as in Subsection 5.1.7. More specifically, given $k \in \mathbb{N}$ and two bounded convex subsets $A$ and $B$ or $\mathbb{R}^2$ with non-empty interior, we are interested in solving grid problems over $\mathbb{Z}[\omega]$ for $\sqrt{2}^k A$ and $(-\sqrt{2})^k B$. We call such problems *scaled grid problems over* $\mathbb{Z}[\omega]$ *for $A$, $B$, and $k$.* These scaled grid problems will also be useful in Chapter 3. Using Proposition 5.2.36, we can establish the following proposition.

**Proposition 5.2.37.** *There is an algorithm which, given two bounded convex subset $A$ and $B$ of $\mathbb{R}^2$ with non-empty interior, enumerates (the infinite sequence of) all solutions of the scaled grid problem over $\mathbb{Z}[\omega]$ for $A$, $B$, and $k$ in order of increasing $k$. Moreover, if $A$ and $B$ are $M$-upright, then the algorithm requires $O(\log(1/M))$ arithmetic operations overall, plus a constant number of arithmetic operations per solution produced.*

Finally, we give some lower bounds on the number of solutions to scaled grid problems over $\mathbb{Z}[\omega]$.

**Lemma 5.2.38.** *Let $A$ and $B$ be convex subsets of $\mathbb{R}^2$, and let $k \geqslant 0$. Assume $A$ contains a circle of radius $r$ and $B$ contains a circle of radius $R$, such that $rR \geqslant \frac{1}{2^k}(1+\sqrt{2})^2$. Then the scaled grid problem over $\mathbb{Z}[\omega]$ for $A$, $B$, and $k$ has at least $2$ solutions.*

*Proof.* By assumption, $(\sqrt{2}^k)A$ contains a circle of radius $r' = \sqrt{2}^k r$ and $(-\sqrt{2})^k B$ contains a circle of radius $R' = \sqrt{2}^k R$, with $rR \geqslant (1+\sqrt{2})^2$. Let $\delta = r'/\sqrt{2}$ and $\Delta = R'\sqrt{2}$, and inscribe two squares of size $\delta \times \delta$ in the first circle, and one square of size $\Delta \times \Delta$ in the second circle, as shown here:



Since $\delta\Delta = r'R' \geqslant (1+\sqrt{2})^2$, by Lemma 5.2.7, we can find $a, a', b \in \mathbb{Z}[\sqrt{2}]$ such that $a \in [x_0, x_1]$, $a^\bullet \in [z_0, z_1]$, $a' \in [x_2, x_3]$, $a'^\bullet \in [z_0, z_1]$, $b \in [y_0, y_1]$, and $b^\bullet \in [w_0, w_1]$. Then $u = a + ib$ and $v = a' + ib$ are two different solutions to the scaled grid problem over $\mathbb{Z}[\omega]$ for $A$, $B$, and $k$ as claimed. □

**Lemma 5.2.39.** *Let $A$ and $B$ be convex subsets of $\mathbb{R}^2$, and assume that the two-dimensional scaled grid problem for $k$ has at least two distinct solutions. Then for all $\ell \geqslant 0$, the scaled grid problem for $k + 2\ell$ has at least $2^\ell + 1$ solutions.*

*Proof.* Analogous to the proof of Proposition 5.1.19. □

# Chapter 6

# Clifford+$V$ approximate synthesis

In this chapter, we introduce an efficient algorithm to solve the problem of approximate synthesis of special unitaries over the Clifford+$V$ gate set. Recall from Chapter 1 that the Clifford group is generated by

$$\omega = e^{i\pi/4}, \quad H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad \text{and} \quad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}.$$

Recall moreover that the Clifford+$V$ group is obtained by adding the following $V$-gates to the generators of the Clifford group

$$V_X = \frac{1}{\sqrt{5}}\begin{bmatrix} 1 & 2i \\ 2i & 1 \end{bmatrix}, \quad V_Y = \frac{1}{\sqrt{5}}\begin{bmatrix} 1 & 2 \\ -2 & 1 \end{bmatrix}, \quad \text{and} \quad V_Z = \frac{1}{\sqrt{5}}\begin{bmatrix} 1+2i & 0 \\ 0 & 1-2i \end{bmatrix}.$$

The problem of approximate synthesis of special unitaries over the Clifford+$V$ gate set is the following. Given a special unitary $U \in \mathrm{SU}(2)$ and a precision $\varepsilon > 0$, construct a Clifford+$V$ circuit $W$ whose $V$-count is as small as possible and such that $\|W - U\| \leqslant \varepsilon$.

We solve the problem in three steps. We first characterize the unitaries which can be expressed *exactly* as Clifford+$V$ circuits. We then use this characterization to define an algorithm solving the problem of approximate synthesis of *$z$-rotations* over the Clifford+$V$ gate set. Finally, we show how to this method can be used to approximately synthesize arbitrary special unitaries.

## 6.1 Exact synthesis of Clifford+$V$ operators

We start by solving the problem of exact synthesis of Clifford+$V$ operators.

**Problem 6.1.1** (Exact synthesis of Clifford+$V$ operators)**.** Given a unitary $U \in \mathrm{U}(2)$, determine whether there exists a Clifford+$V$ circuit $W$ such that $U = W$ and, in case such a circuit exists, construct one whose $V$-count is minimal.

The problem of exact synthesis of *Pauli+V* operators was first solved in [7] using the arithmetic of quaternions. Here, we extend this result to the Clifford+$V$ group.

To characterize Clifford+$V$ operators, we consider the following set of unitaries.

**Definition 6.1.2.** The set $\mathcal{V}$ consists of unitary matrices of the form

$$U = \frac{1}{\sqrt{2}^k\sqrt{5}^\ell}\begin{bmatrix} \alpha & \gamma \\ \beta & \delta \end{bmatrix} \tag{6.1}$$

where $k, \ell \in \mathbb{N}$ with $0 \leqslant k \leqslant 2$, $\alpha, \beta, \gamma, \delta \in \mathbb{Z}[i]$, and such that $\det(U)$ is a power of $i$.

We will show that a unitary $U$ is a Clifford+$V$ operator if and only if $U \in \mathcal{V}$. As a corollary, this will establish that $\mathcal{V}$ is a group, which might not be obvious, due to the seemingly arbitrary restriction on the exponent $k$. We note that $\mathcal{V}$ does not coincide with the subgroup of $\mathrm{U}(2)$ whose entries are in the ring $\mathbb{Z}[1/\sqrt{2}, 1/\sqrt{5}, i]$ and whose determinant is a power of $i$. Indeed, the matrix

$$\frac{1}{5^3}\begin{bmatrix} 2i+\sqrt{5} & -80+96i \\ 80+96i & -2i+\sqrt{5} \end{bmatrix}$$

has entries in $\mathbb{Z}[1/\sqrt{2}, 1/\sqrt{5}, i]$ and has determinant 1 but is not an element of $\mathcal{V}$.

**Definition 6.1.3.** Let $U \in \mathcal{V}$ be as in (6.1). The integers $k$ and $\ell$ are called the $\sqrt{2}$-*denominator exponent* and $\sqrt{5}$-*denominator exponent* of $U$ respectively. The least $k$ (resp. $\ell$) such that $U$ can be written as in (6.1) is the *least* $\sqrt{2}$-*denominator exponent* (resp. *least* $\sqrt{5}$-*denominator exponent*) of $U$. These notions extend naturally to vectors and scalars of the form

$$\frac{1}{\sqrt{2}^k\sqrt{5}^\ell}\begin{bmatrix} \alpha \\ \beta \end{bmatrix} \quad \text{and} \quad \frac{1}{\sqrt{2}^k\sqrt{5}^\ell}\,\alpha, \tag{6.2}$$

where $k, \ell \in \mathbb{N}$, with $0 \leqslant k \leqslant 2$, and $\alpha, \beta \in \mathbb{Z}[i]$.

In what follows, we refer to the pair $(k, \ell)$ as the *denominator exponent* of a matrix, vector, or scalar. It is then understood that the first component of the pair $(k, \ell)$ is the $\sqrt{2}$-exponent, while the second is the $\sqrt{5}$-exponent. Note that the least denominator exponent of a matrix, vector, or scalar is the pair $(k, \ell)$, where $k$ and $\ell$ are the least $\sqrt{2}$- and $\sqrt{5}$-exponents respectively.

*Remark* 6.1.4. Since $\sqrt{5} \notin \mathbb{Z}[i]$, if $\ell$ and $\ell'$ are two $\sqrt{5}$-denominator exponents of a matrix $U \in \mathcal{V}$, then $\ell \equiv \ell' \pmod 2$. A similar property holds for $\sqrt{2}$-denominator exponents.

We first show that if $U$ is a Clifford+$V$ operator, then $U \in \mathcal{V}$.

**Lemma 6.1.5.** *If $U$ is a Clifford+$V$ operator, then $U = ABC$ where $A$ is a product of $V$-gates, $B$ is a Pauli+$S$ operator, and $C$ is one of $I$, $H$, $HS$, $\omega$, $H\omega$, and $HS\omega$.*

*Proof.* Clifford gates and $V$-gates can be commuted in the sense that for every pair $C, V$ of a Clifford gate and a $V$-gate, there exists a pair $C', V'$ such that $CV = V'C'$. This implies that a Clifford+$V$ operator $U$ can always be written as $U = AA'$, where $A$ is a product of $V$-gates and $A'$ is a Clifford operator. Furthermore, the Pauli+$S$ group has index 6 as a subgroup of the Clifford group and its cosets are: Pauli+$S$, Pauli+$S \cdot H$, Pauli+$S \cdot HS$, Pauli+$S \cdot \omega$, Pauli+$S \cdot H\omega$, and Pauli+$S \cdot HS\omega$. It thus follows that a Clifford operator $A'$ can always be written as $A' = BC$ with $B$ a Pauli+$S$ operator and $C$ one of $I$, $H$, $HS$, $\omega$, $H\omega$, and $HS\omega$. □

**Corollary 6.1.6.** *If $U$ is a Clifford+$V$ operator, then $U \in \mathcal{V}$.*

To show, conversely, that every element of $\mathcal{V}$ can be represented by a Clifford+$V$ circuit, we proceed as follows. First, we show that every unit vector of the form (6.2) can be reduced to $e_1 = \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$ by applying a sequence of carefully chosen Clifford+$V$ gates. Then, we show how applying this method to the first column of a unitary matrix $U$ of the form (6.1) yields a Clifford+$V$ circuit for $U$.

**Lemma 6.1.7.** *If $u$ is a unit vector of the form (6.2) with least $\sqrt{5}$-denominator exponent $\ell$ and $W$ is a Clifford circuit, then $Wu$ has least $\sqrt{5}$-denominator exponent $\ell$.*

*Proof.* It suffices to show that the generators of the Clifford group preserve the least $\sqrt{5}$-denominator exponent of $u$. The general result then follows by induction. To this end, write $u$ as in (6.2), with $\alpha = a + ib$ and $\beta = c + id$:

$$u = \frac{1}{\sqrt{2}^k \sqrt{5}^\ell} \begin{bmatrix} a + ib \\ c + id \end{bmatrix}.$$

Now apply $H$, $\omega$, and $S$ to $u$:

$$Hu = \frac{1}{\sqrt{2}^{k+1}\sqrt{5}^{\ell}}\begin{bmatrix}(a+c)+i(b+d)\\(a-c)+i(b-d)\end{bmatrix}, \quad \omega u = \frac{1}{\sqrt{2}^{k+1}\sqrt{5}^{\ell}}\begin{bmatrix}(a-b)+i(a+b)\\(c-d)+i(c+d)\end{bmatrix},$$

$$Su = \frac{1}{\sqrt{2}^{k}\sqrt{5}^{\ell}}\begin{bmatrix}a+ib\\-d+ic\end{bmatrix}.$$

By minimality of $\ell$, one of $a, b, c, d$ is not divisible by 5. The least $\sqrt{5}$-denominator of $Su$ is therefore $\ell$. Moreover, for any two integers $x$ and $y$, $x+y \equiv x-y \equiv 0 \pmod 5$ implies $x \equiv y \equiv 0 \pmod 5$. Thus the least $\sqrt{5}$-denominator exponent of $Hu$ and $\omega u$ is also $\ell$. □

**Lemma 6.1.8.** *If $u$ is a unit vector of the form (6.2) with least denominator exponent $(k, \ell)$, then there exists a Clifford circuit $W$ such that $Wu$ has least denominator exponent $(0, \ell)$.*

*Proof.* By Lemma 6.1.7, we need not worry about $\ell$ and only have to focus on reducing $k$. Write $u$ as in (6.2), with $0 \leqslant k \leqslant 2$, $\alpha = a+ib$, and $\beta = c+id$. Since $u$ has unit norm, we have $a^2+b^2+c^2+d^2 = 2^k \cdot 5^\ell$. We prove the lemma by case distinction on $k$. If $k=0$, there is nothing to prove. The remaining cases are treated as follows.

- $k=1$. In this case $a^2+b^2+c^2+d^2 = 2\cdot 5^\ell \equiv 0 \pmod 2$. Therefore only an even number amongst $a, b, c, d$ can be odd. Using a Pauli+$S$ operator, we can without loss of generality assume that $a \equiv c \pmod 2$ and $b \equiv d \pmod 2$ or that $a \equiv b \pmod 2$ and $c \equiv d \pmod 2$. It then follows that either $Hu$ or $\omega u$ has denominator exponent $(0, \ell)$ since

$$Hu = \frac{1}{2\sqrt{5}^{\ell}}\begin{bmatrix}(a+c)+i(b+d)\\(a-c)+i(b-d)\end{bmatrix} \quad \text{and} \quad \omega u = \frac{1}{2\sqrt{5}^{\ell}}\begin{bmatrix}(a-b)+i(a+b)\\(c-d)+i(c+d)\end{bmatrix}.$$

- $k=2$. In this case $a^2+b^2+c^2+d^2 = 4\cdot 5^\ell \equiv 0 \pmod 4$. This implies that $a, b, c$ and $d$ must have the same parity and thus, by minimality of $k$, must all be odd. Using a Pauli+$S$ operator, we can without loss of generality assume that $a \equiv b \equiv c \equiv d \equiv 1 \pmod 4$. It then follows that $H\omega u$ has denominator exponent $(0, \ell)$ since

$$H\omega u = \frac{1}{4\sqrt{5}^{\ell}}\begin{bmatrix}(a-b+c-d)+i(a+b+c+d)\\(a-b-c+d)+i(a+b-c-d)\end{bmatrix}.$$

□

*Remark* 6.1.9. Let $V$ be one of the $V$-gates, $u$ be a vector of the form (6.2), and $\ell$ and $\ell'$ be the least $\sqrt{5}$-denominator exponents of $u$ and $Vu$ respectively. Then $\ell' \leqslant \ell + 1$. Moreover, if it were the case that $\ell' < \ell - 1$, then the least $\sqrt{5}$-denominator exponent of $V^{\dagger}Vu = u$ would be strictly less $\ell$ which is absurd. Thus $\ell - 1 \leqslant \ell' \leqslant \ell + 1$.

**Lemma 6.1.10.** *If $u$ is a unit vector of the form (6.2) with least denominator exponent $(0, \ell)$, then there exists a Pauli+V circuit $W$ of V-count $\ell$ such that $Wu = e_1$, the first standard basis vector.*

*Proof.* Write $u$ as in (6.2) with $k = 0$, $\alpha = a + ib$, and $\beta = c + id$. Since $u$ has unit norm, we have $a^2 + b^2 + c^2 + d^2 = 2^0 \cdot 5^{\ell} = 5^{\ell}$. We prove the lemma by induction on $\ell$.

- $\ell = 0$. In this case $a^2 + b^2 + c^2 + d^2 = 1$. It follows that exactly one of $a, b, c, d$ is $\pm 1$ while all the others are 0. Then $u$ can be reduced to $e_1$ by acting on it using a Pauli operator.

- $\ell > 0$. In this case $a^2 + b^2 + c^2 + d^2 \equiv 0 \pmod 5$. We will show that there exists a Pauli+$V$ operator $U$ of $V$-count 1 such that the least denominator exponent of $Uu$ is $\ell - 1$. It then follows by the induction hypothesis that there exists $U'$ of $V$-count $\ell - 1$ such that $U'Uu = e_1$, which then completes the proof.

  Consider the residues modulo 5 of $a, b, c,$ and $d$. Since $0, 1,$ and $4$ are the only squares modulo 5, then, up to a reordering of the tuple $(a, b, c, d)$, we must have:

$$(a, b, c, d) \equiv \begin{cases} (0, 0, 0, 0) \\ (\pm 2, \pm 1, 0, 0) \\ (\pm 2, \pm 2, \pm 1, \pm 1). \end{cases}$$

  However, by minimality of $\ell$, we know that $a \equiv b \equiv c \equiv d \equiv 0$ is impossible, so the other two cases are the only possible ones. We treat them in turn.

  First, assume that one of $a, b, c, d$ is congruent to $\pm 2$, one is congruent to $\pm 1$, and the remaining two are congruent to 0. By acting on $u$ with a Pauli operator, we can moreover assume without loss of generality that $a \equiv 2$. Now if $b \equiv 1$, consider $V_Z u$:

$$V_Z u = \frac{1}{\sqrt{5}^{k+1}} \begin{bmatrix} (a - 2b) + i(2a + b) \\ (c + 2d) + i(d - 2c) \end{bmatrix}.$$

Since $a \equiv 2$, $b \equiv 1$, and $c \equiv d \equiv 0$, we get $(a - 2b) \equiv (2a + b) \equiv (c + 2d) \equiv (d - 2c) \equiv 0 \pmod 5$. The least denominator exponent of $V_Z u$ is therefore $\ell - 1$. If on the other hand $b \equiv -1$ then

$$V_Z{}^{\dagger} u = \frac{1}{\sqrt{5}^{k+1}} \begin{bmatrix} (a + 2b) + i(b - 2a) \\ (c - 2d) + i(d + 2c) \end{bmatrix}$$

and reasoning analogously shows that the least denominator exponent of $V_Z{}^{\dagger} u$ is $k - 1$. A similar argument can be made in the remaining cases, i.e., when $c \equiv \pm 1$ or $d \equiv \pm 1$. For brevity, we list the desired operators in the table below. The left column describes the residues of $a, b, c,$ and $d$ modulo 5 and the right column gives the operator $U$ such that $Uu$ has least denominator exponent $\ell - 1$.

| $(a, b, c, d)$ | $U$ |
|---:|:---|
| $(2, 1, 0, 0)$ | $V_Z$ |
| $(2, 0, 1, 0)$ | $V_Y{}^{\dagger}$ |
| $(2, 0, 0, 1)$ | $V_X$ |
| $(2, -1, 0, 0)$ | $V_Z^{\dagger}$ |
| $(2, 0, -1, 0)$ | $V_Y$ |
| $(2, 0, 0, -1)$ | $V_X{}^{\dagger}$ |

Now assume that two of $a, b, c, d$ are congruent to $\pm 2$ while the remaining two are congruent to $\pm 1$. We can use Pauli operators to guarantee that $a \equiv 2$ and $c \geqslant 0$. As above, we list the desired operators in a table for conciseness. It can be checked that in each case the given operator is such that the least denominator exponent of $Uu$ is $\ell - 1$.

| $(a,b,c,d)$ | $U$ |
|---|---|
| $(2,2,1,1)$ | ${V_Y}^\dagger$ |
| $(2,1,2,1)$ | $V_X$ |
| $(2,1,1,2)$ | $V_Z$ |
| $(2,1,2,-1)$ | $V_Z$ |
| $(2,-1,2,1)$ | ${V_Z}^\dagger$ |
| $(2,2,1,-1)$ | ${V_X}^\dagger$ |
| $(2,-2,1,1)$ | $V_X$ |
| $(2,1,1,-2)$ | ${V_Y}^\dagger$ |
| $(2,-1,1,2)$ | ${V_Y}^\dagger$ |
| $(2,-1,1,-2)$ | ${V_Z}^\dagger$ |
| $(2,-1,2,-1)$ | ${V_X}^\dagger$ |
| $(2,-2,1,-1)$ | ${V_Y}^\dagger$ |

□

We can now solve Problem 6.1.1.

**Proposition 6.1.11.** *A unitary operator $U \in U(2)$ is exactly representable by a Clifford+V circuit if and only if $U \in \mathcal{V}$. Moreover, there exists an efficient algorithm that computes a Clifford+V circuit for $U$ with V-count equal to the least $\sqrt{5}$-denominator exponent of $U$, which is minimal.*

*Proof.* The left-to-right implication is given by Corollary 6.1.6. For the right-to-left implication, it suffices to show that there exists a Clifford+$V$ circuit $W$ of $V$-count $\ell$ such that $WU = I$, since we then have $U = W^\dagger$. To construct $W$, apply Lemma 6.1.8 and Lemma 6.1.10 to the first column $u_1$ of $U$. This yields a circuit $W'$ such that the first column of $W'U$ is $e_1$. Since $W'U$ is unitary, it follows that its second column $u_2$ is a unit vector orthogonal to $e_1$. Therefore $u_2 = \lambda e_2$ where $\lambda$ is a unit of the Gaussian integers. Since the determinant of $W'$ is $i^m$ for some integer $m$, the determinant of $W'U$ is $i^{n+m}$, so that $\lambda = i^{n+m}$. Thus one of the following equalities must hold

$$W'U = I,\ ZW'U = I,\ SW'U = I \text{ or } ZSW'U = I.$$

To prove the second claim, suppose that the least $\sqrt{5}$-denominator exponent of $U$ is $\ell$. Then $W$ can be efficiently computed because the algorithm described in the proofs of Lemma 6.1.8 and Lemma 6.1.10 requires $O(\ell)$ arithmetic operations. Moreover, $W$ has $V$-count $\ell$ by Lemma 6.1.10, which is minimal since any Clifford+$V$ circuit of $V$-count up to $\ell - 1$ has least $\sqrt{5}$-denominator exponent at most $\ell - 1$. □

*Remark* 6.1.12. By restricting $k$ to be equal to 0 in (6.1) and the determinant of $U$ to be ±1 we get a solution to the problem of exact synthesis for Pauli+$V$ operators.

## 6.2 Approximate synthesis of $z$-rotations

We now turn to the approximate synthesis of *$z$-rotations* over the Clifford+$V$ gate set. A $z$-rotation is a unitary matrix of the form

$$R_z(\theta) = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix} \tag{6.3}$$

for some real number $\theta$. Matrices of the form (6.3) are called $z$-rotations because they act as rotations of the Bloch sphere along the $z$-axis.

**Problem 6.2.1.** Given an angle $\theta$ and a precision $\varepsilon > 0$, construct a Clifford+$V$ circuit $U$ whose $V$-count is as small as possible and such that $\|U - R_z(\theta)\| \leqslant \varepsilon$.

### 6.2.1 A reduction of the problem

**Lemma 6.2.2.** *Let $c_V = |1 - e^{i\pi/4}|$. If $\varepsilon < c_V$, then all solutions to Problem 6.2.1 have determinant 1. If $\varepsilon \geqslant c_V$, then there exists a solution of the form $\omega^n$ for some $n \in \mathbb{N}$.*

*Proof.* Every complex $2 \times 2$ unitary operator $U$ can be written as

$$U = \begin{bmatrix} a & -b^\dagger e^{i\phi} \\ b & a^\dagger e^{i\phi} \end{bmatrix},$$

for $a, b \in \mathbb{C}$ and $\phi \in [-\pi, \pi]$. This, together with the characterization of Clifford+$V$ operators given by Proposition 6.1.11, implies that a complex $2 \times 2$ unitary operator $U$ can be exactly synthesized over the Clifford+$V$ gate set if and only if

$$U = \frac{1}{\sqrt{2}^k \sqrt{5}^\ell} \begin{bmatrix} \alpha & -\beta^\dagger i^n \\ \beta & \alpha^\dagger i^n \end{bmatrix},$$

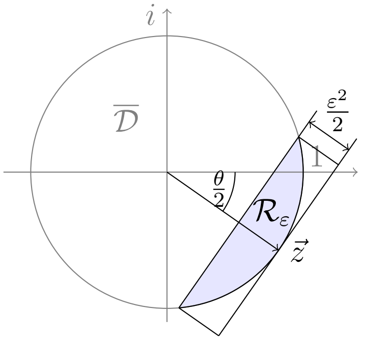


Figure 6.1: The $\varepsilon$-region..

with $k, \ell, n \in \mathbb{N}$, $\alpha, \beta \in \mathbb{Z}[i]$, and $0 \leqslant \ell \leqslant 2$. Now assume that $\varepsilon < |1 - e^{i\pi/4}|$ and $\|U - R_z(\theta)\| \leqslant \varepsilon$. Let $e^{i\phi_1}$ and $e^{i\phi_2}$ be the eigenvalues of $UR_z(\theta)^{-1}$, with $\phi_1, \phi_2 \in [-\pi, \pi]$. Then $|1 - e^{i\pi/4}| > \varepsilon \geqslant \|U - R_z(\theta)\| = \|I - UR_z(\theta)^{-1}\| = \max\{|1 - e^{i\phi_1}|, |1 - e^{i\phi_2}|\}$, so that $|1 - e^{i\phi_j}| < |1 - e^{i\pi/4}|$. Therefore $-\pi/4 < \phi_j < \pi/4$, for $j \in \{1, 2\}$, which implies that $-\pi/2 < \phi_1 + \phi_2 < \pi/2$. Hence $|1 - e^{i(\phi_1+\phi_2)}| < |1 - e^{i\pi/2}| = \sqrt{2}$. But $e^{i(\phi_1+\phi_2)} = \det(UR_z(\theta)^{-1}) = i^n$. Thus $|1 - i^n| < \sqrt{2}$ which proves that $i^n = 1$.

For the second statement, note that if $\theta/2 \in [-\pi/4, \pi/4]$, then $\|I - R_z(\theta)\| = |1 - e^{i\theta/2}| \leqslant |1 - e^{i\pi/4}|$. Similarly, if $\theta/2$ belongs to one of $[\pi/4, 3\pi/4]$, $[3\pi/4, 5\pi/4]$, or $[5\pi/4, 7\pi/4]$, then one of $\|\omega^2 - R_z(\theta)\|$, $\| - I - R_z(\theta)\|$, or $\| - \omega^2 - R_z(\theta)\|$ is less than $|1 - e^{i\pi/4}|$. In each case, $R_z(\theta)$ is approximated to within $\varepsilon$ by a Clifford operator. □

**Definition 6.2.3.** Let $\theta$ be an angle and $\varepsilon > 0$ a precision. The *$\varepsilon$-region for $\theta$* is the subset of the plane defined by

$$\mathcal{R}_\varepsilon = \{u \in \overline{\mathcal{D}} \ ; \ u \cdot z \geqslant 1 - \frac{\varepsilon^2}{2}\}$$

where $z = e^{-i\theta/2}$ and $\overline{\mathcal{D}}$ is the unit disk.

The $\varepsilon$-region is illustrated in Figure 6.1. We now show how to reduce Problem 6.2.1 to three distinct problems.

**Proposition 6.2.4.** *Problem 6.2.1 reduces to a grid problem, a Diophantine equation, and an exact synthesis problem, namely:*

1. *find $k, \ell \in \mathbb{N}$ with $0 \leqslant k \leqslant 2$ and $\alpha \in \mathbb{Z}[i]$ such that $\alpha \in \sqrt{2}^k\sqrt{5}^\ell\mathcal{R}_\varepsilon$,*

*2. find $\beta \in \mathbb{Z}[i]$ such that $\beta^\dagger\beta = 2^k 5^\ell - \alpha^\dagger\alpha$, and*

*3. find a Clifford+V circuit for the unitary matrix*

$$U = \frac{1}{\sqrt{2}^k\sqrt{5}^\ell}\begin{bmatrix}\alpha & -\beta^\dagger \\ \beta & \alpha^\dagger\end{bmatrix}.$$

*Moreover, the least $\sqrt{2}^k\sqrt{5}^\ell$ for which the above three problems can be solved yields an optimal solution to Problem 6.2.1.*

*Proof.* Let $U$ be the matrix

$$U = \frac{1}{\sqrt{2}^k\sqrt{5}^\ell}\begin{bmatrix}\alpha & -\beta^\dagger \\ \beta & \alpha^\dagger\end{bmatrix} \tag{6.4}$$

with $\alpha, \beta \in \mathbb{Z}[i]$ and $k, \ell \in \mathbb{N}$ satisfying $0 \leqslant k \leqslant 2$ and $\alpha^\dagger\alpha = 2^k 5^\ell$. Given $\varepsilon$ and $\theta$, we can express the requirement $\|U - R_z(\theta)\| \leqslant \varepsilon$ as a constraint on the top left entry $\alpha/(\sqrt{5}^k\sqrt{2}^\ell)$ of $U$. Indeed, let $z = e^{-i\theta/2}$, $\alpha' = \alpha/(\sqrt{5}^k\sqrt{2}^\ell)$, and $\beta' = \beta/(\sqrt{5}^k\sqrt{2}^\ell)$. Since ${\alpha'}^\dagger\alpha' + {\beta'}^\dagger\beta' = 1$ and $z^\dagger z = 1$, we have

$$\begin{aligned}\|U - R_z(\theta)\|^2 &= |\alpha' - z|^2 + |\beta'|^2 \\ &= (\alpha' - z)^\dagger(\alpha' - z) + {\beta'}^\dagger\beta' \\ &= {\alpha'}^\dagger\alpha' + {\beta'}^\dagger\beta' - z^\dagger\alpha' - {\alpha'}^\dagger z + z^\dagger z \\ &= 2 - 2\operatorname{Re}(z^\dagger\alpha').\end{aligned}$$

Thus $\|R_z(\theta) - U\| \leqslant \varepsilon$ if and only if $2 - 2\operatorname{Re}(z^\dagger\alpha') \leqslant \varepsilon^2$, or equivalently, $\operatorname{Re}(z^\dagger\alpha') \geqslant 1 - \frac{\varepsilon^2}{2}$. If we identify the complex numbers $z = x + yi$ and $\alpha' = a + bi$ with 2-dimensional real vectors $\vec{z} = (x, y)^T$ and $\vec{\alpha}' = (a, b)^T$, then $\operatorname{Re}(z^\dagger\alpha')$ is just their inner product $\vec{z} \cdot \vec{\alpha}'$, and therefore $\|U - R_z(\theta)\| \leqslant \varepsilon$ is equivalent to $\vec{z} \cdot \vec{\alpha}' \geqslant 1 - \varepsilon^2/2$. Hence

$$\|U - R_z(\theta)\| \leqslant \varepsilon \Longleftrightarrow \alpha \in \sqrt{2}^k\sqrt{5}^\ell\mathcal{R}_\varepsilon.$$

The fact that, by Lemma 6.2.2, all the solutions to Problem 6.2.1 are of the form (6.4) completes the reduction. Since by Proposition 6.1.11 the minimal $V$-count of an element $U \in \mathcal{V}$ is its least $\sqrt{5}$-denominator exponent, then the least $\sqrt{2}^k\sqrt{5}^\ell$ for which problems 1-3 can be solved is an optimal solution. □

### 6.2.2 The algorithm

The reduction of Proposition 6.2.4 describes an algorithm which we explicitly state below.

**Algorithm 6.2.5.** Let $\theta$ and $\varepsilon > 0$ be given.

1. Use the algorithm from Proposition 5.1.17 of Chapter 5 to enumerate the infinite sequence of solutions to the scaled grid problem over $\mathbb{Z}[i]$ for $\mathcal{R}_\varepsilon$ and $\sqrt{2}^k\sqrt{5}^\ell$ in order of increasing $\ell$.

2. For each solution $\alpha$:

   (a) Let $n = 2^k 5^\ell - \alpha^\dagger\alpha$.

   (b) Attempt to find a prime factorization of $n$. If $n \neq 0$ but no prime factorization is found, skip step 2(c) and continue with the next $\alpha$.

   (c) Use the algorithm from Proposition 3.2.8 of Chapter 3 to solve the equation $\beta^\dagger\beta = n$. If a solution $\beta$ exists, go to step 3; otherwise, continue with the next $\alpha$.

3. Define $U$ as
$$U = \frac{1}{\sqrt{2}^k\sqrt{5}^\ell}\begin{bmatrix} \alpha & -\beta^\dagger \\ \beta & \alpha^\dagger \end{bmatrix}$$
and use the exact synthesis algorithm of Proposition 6.1.11 to find a Clifford+$V$ circuit for $U$. Output this circuit and stop.

*Remark* 6.2.6. In analogy with Remark 6.1.12, we note that restricting $k$ to be equal to 0 throughout the algorithm yields a method for the approximate synthesis of $z$-rotations in the Pauli+$V$ basis.

### 6.2.3 Analysis of the algorithm

We now discuss the properties of Algorithm 6.2.5. We are interested in three aspects of the algorithm: its correctness, its circuit complexity, and its time complexity. We treat each of these aspects in turn. We note that the results established here also hold for the restricted algorithm of Remark 6.2.6.

### Correctness

**Proposition 6.2.7** (Correctness). *If Algorithm 6.2.5 terminates, then it yields a valid solution to the approximate synthesis problem, i.e., it yields a Clifford+V circuit approximating $R_z(\theta)$ up to $\varepsilon$.*

*Proof.* By construction, following the reduction described by Proposition 6.2.4. □

### Circuit complexity

In the presence of a factoring oracle, the algorithm has optimal circuit complexity.

**Proposition 6.2.8** (Optimality in the presence of a factoring oracle). *In the presence of an oracle for integer factoring, the circuit returned by Algorithm 6.2.5 has the smallest V-count of any single-qubit Clifford+V circuit approximating $R_z(\theta)$ up to $\varepsilon$.*

*Proof.* By construction, step (1) of the algorithm enumerates all solutions $\alpha$ to the scaled grid problem over $\mathbb{Z}[i]$ for $\mathcal{R}_\varepsilon$ and $\sqrt{2}^k\sqrt{5}^\ell$ in order of increasing $\ell$. Step 2(a) always succeeds and, in the presence of the factoring oracle, so does step 2(b). When step 2(c) succeeds, the algorithm has found a solution of Problem 6.2.1 of minimal $\sqrt{5}$-denominator exponent, which therefore has minimal $V$-count. □

In the absence of a factoring oracle, the algorithm is still nearly optimal. Our proof of this near-optimality relies on the following number-theoretic hypothesis. We do not have a proof of this hypothesis, but it appears to be valid in practice.

**Hypothesis 6.2.9.** For each number $n$ produced in step 2(a) of Algorithm 6.2.5, write $n = 2^j m$, where $m$ is odd. Then $m$ is asymptotically as likely to be a prime congruent to 1 modulo 4 as a randomly chosen odd number of comparable size. Moreover, each $m$ can be modelled as an independent random event.

**Definition 6.2.10.** Let $U'$ and $U''$ be the following two solutions of the approximate synthesis problem

$$U' = \begin{bmatrix} \alpha' & -\beta'^\dagger \\ \beta' & \alpha'^\dagger \end{bmatrix} \quad \text{and} \quad U'' = \begin{bmatrix} \alpha'' & -\beta''^\dagger \\ \beta'' & \alpha''^\dagger \end{bmatrix}. \tag{6.5}$$

$U'$ and $U''$ are said to be *equivalent solutions* if $\alpha' = \alpha''$.

**Proposition 6.2.11** (Near-optimality in the absence of a factoring oracle)**.** *Let $\ell$ be the V-count of the solution of the approximate synthesis problem found by Algorithm 6.2.5 in the absence of a factoring oracle. Then*

1. *The approximate synthesis problem has at most $O(\log(1/\varepsilon))$ non-equivalent solutions with V-count less than $\ell$.*
2. *The expected value of $\ell$ is $\ell''' + O(\log(\log(1/\varepsilon)))$, where $\ell', \ell''$, and $\ell'''$ are the V-counts of the optimal, second-to-optimal, and third-to-optimal solutions of the approximate synthesis problem (up to equivalence).*

*Proof.* If $\varepsilon \geqslant |1 - e^{i\pi/4}|$, then by Lemma 6.2.2 there is a solution of $V$-count 0 and the algorithm easily finds it. In this case there is nothing to show, so assume without loss of generality that $\varepsilon < |1 - e^{i\pi/4}|$. Then by Lemma 6.2.2, all solutions are of the form (6.4).

1. Consider the list $\alpha_1, \alpha_2, \ldots$ of candidates generated in step (i) of the algorithm. Let $\ell_1, \ell_2, \ldots$ be their least $\sqrt{5}$-denominator exponent and let $n_1, n_2, \ldots$ be the corresponding integers calculated in step (ii.a). Note that $n_j \leqslant 4 \cdot 5^{\ell_j}$ for all $j$. Write $n_j = 2^{z_j} m_j$ where $m_j$ is odd. By Hypothesis 6.2.9, the probability that $m_j$ is a prime congruent to 1 modulo 4 is asymptotically no smaller than that of a randomly chosen odd integer less than $4 \cdot 5^{\ell_j}$, which, by the well-known prime number theorem, is
$$p_j := \frac{1}{\ln(4 \cdot 5^{\ell_j})} = \frac{1}{\ell_j \ln 5 + \ln 4}. \tag{6.6}$$
By the pigeon-hole principle, two of $\ell_1, \ell_2$, and $\ell_3$ must be congruent modulo 2. Assume without loss of generality that $\ell_2 \equiv \ell_3 \pmod 2$. Then $\alpha_2$ and $\alpha_3$ are two distinct solutions to the scaled grid problem over $\mathbb{Z}[i]$ for $\mathcal{R}_\varepsilon$ and $\sqrt{2}^k\sqrt{5}^\ell$ with (not necessarily least) $\sqrt{5}$-denominator exponent $\ell_3$. It follows by Proposition 5.1.19 from Chapter 5 that there are at least $5^r + 1$ distinct candidates of denominator exponent $\ell_3 + 2r$, for all $r \geqslant 0$. In other words, for all $j$, if $j \leqslant 5^r + 1$, we have $\ell_j \leqslant \ell_3 + 2r$. In particular, this holds for $r = \lfloor 1 + \log_5 j \rfloor$, and therefore,
$$\ell_j \leqslant \ell_3 + 2(1 + \log_5 j). \tag{6.7}$$

Combining (6.7) with (6.6), we have

$$p_j \geqslant \frac{1}{(\ell_3 + 2(1+\log_5 j))\ln 5 + \ln 4} = \frac{1}{(\ell_3+2)\ln 5 + 2\ln j + \ln 4} \tag{6.8}$$

Let $j_0$ be the smallest index such that $m_{j_0}$ is a prime congruent to 1 modulo 4. By Hypothesis 6.2.9, we can treat each $m_j$ as an independent random variable. Therefore,

$$\begin{aligned} P(j_0 > j) &= P(n_1, \ldots, n_j \text{ are not prime}) \\ &\leqslant (1-p_1)(1-p_2)\cdots(1-p_j) \\ &\leqslant (1-p_j)^j \\ &\leqslant \left(1 - \frac{1}{(\ell_3+2)\ln 5 + 2\ln j + \ln 4}\right)^j. \end{aligned}$$

The expected value of $j_0$ is given by the sum $E(j_0) = \sum_{j=0}^{\infty} P(j_0 > j)$. It was proved in [56] that this sum can be estimated as follows

$$\begin{aligned} E(j_0) &= \sum_{j=0}^{\infty} P(j_0 > j) \\ &\leqslant 1 + \sum_{j=1}^{\infty} \left(1 - \frac{1}{(\ell_3+2)\ln 5 + 2\ln j + \ln 4}\right)^j = O(\ell_3). \end{aligned} \tag{6.9}$$

Next, we will estimate $\ell_3$. First note that if the $\varepsilon$ region contains a circle of radius greater than $1/\sqrt{5}^{\ell}$, then it contains at least 3 solutions to the scaled grid problem for $\mathcal{R}_\varepsilon$ with $\sqrt{5}$-denominator exponent $\ell$. The width of the $\varepsilon$-region $\mathcal{R}_\varepsilon$ is $\varepsilon^2/2$ at the widest point, and we can inscribe a disk of radius $r = \varepsilon^2/4$ in it. Hence the scaled grid problem over $\mathbb{Z}[i]$ for $\mathcal{R}_\varepsilon$, as in step 1 of the algorithm, has at least three solutions with denominator exponent $\ell$, provided that

$$r = \frac{\varepsilon^2}{4} \geqslant \frac{1}{\sqrt{5}^{\ell}},$$

or equivalently, provided that

$$\ell \geqslant 2\log_5(2) + 2\log_5(1/\varepsilon).$$

It follows that

$$\ell_3 = O(\log(1/\varepsilon)), \tag{6.10}$$

and therefore, using (6.9), also

$$E(j_0) = O(\log(1/\varepsilon)). \tag{6.11}$$

To finish the proof of part (i), recall that $j_0$ was defined to be the smallest index such that $m_{j_0}$ is a prime congruent to 1 modulo 4. The primality of $m_{j_0}$ ensures that step (ii.b) of the algorithm succeeds for the candidate $\alpha_{j_0}$. Furthermore, because $m_{j_0} \equiv 1 \pmod 4$, the equation $\beta^\dagger\beta = n$ has a solution by Proposition 3.2.6. Hence the remaining steps of the algorithm also succeed for $\alpha_{j_0}$.

Now let $s$ be the number of non-equivalent solutions of the approximate synthesis problem of $V$-count strictly less than $\ell$. As noted above, any such solution $U$ is of the form (6.4). Then the least denominator exponent of $\alpha$ is strictly smaller than $\ell_{j_0}$, so that $\alpha = \alpha_j$ for some $j < j_0$. In this way, each of the $s$ non-equivalent solutions is mapped to a different index $j < j_0$. It follows that $s < j_0$, and hence that $E(s) \leqslant E(j_0) = O(\log(1/\varepsilon))$, as was to be shown.

2. Let $U'$ be an optimal solution of the approximate synthesis problem, let $U''$ be optimal among the solutions that are not equivalent to $U'$ and let $U'''$ be optimal among the solutions that are not equivalent to either $U'$ or $U''$. Assume that $U', U''$, and $U'''$ are written as in (6.5) with top-left entry $\alpha', \alpha''$, and $\alpha'''$ respectively. Now let $\ell'$, $\ell''$, and $\ell'''$ be the least denominator exponents of $\alpha'$, $\alpha''$, and $\alpha'''$, respectively. Let $\ell_3$ and $j_0$ be as in the proof of part (i). Note that, by definition, $\ell_3 \leqslant \ell'''$. Let $\ell$ be the least denominator exponent of the solution of the approximate synthesis problem found by the algorithm. Then $\ell \leqslant \ell_{j_0}$. Using (6.7), we have

$$\ell \leqslant \ell_{j_0} \leqslant \ell_3 + 2(1 + \log_5 j_0) \leqslant \ell''' + 2(1 + \log_5 j_0).$$

This calculation applies to any one run of the algorithm. Taking expected values over many randomized runs, we therefore have

$$E(\ell) \leqslant \ell''' + 2 + 2E(\log_5 j_0) \leqslant \ell''' + 2 + 2\log_5 E(j_0). \tag{6.12}$$

Note that we have used the law $E(\log j_0) \leqslant \log(E(j_0))$, which holds because log is a concave function. Combining (6.12) with (6.11), we therefore have the desired result:

$$E(\ell) = \ell''' + O(\log(\log(1/\varepsilon))).$$

□

### Time complexity

Finally, we turn to the time complexity of the algorithm. For this again, we rely on our number theoretic conjecture on the distribution of primes to estimate how many candidates must be tried before one that is prime is reached.

**Proposition 6.2.12.** *Algorithm 6.2.5 runs in expected time* $O(\text{polylog}(1/\varepsilon))$. *This is true whether or not a factorization oracle is used.*

*Proof.* Let $M$ be the uprightness of the $\varepsilon$-region. Let $j_0$ be the average number of candidates tried in steps 2(a)–(c) of the algorithm, and let $\ell_{j_0}$ be the least denominator exponent of the final candidate. Let $n$ be the largest integer that appears in step 2(a) of the algorithm.

By Proposition 5.1.17, step 1 of the algorithm requires $O(\log(1/M))$ arithmetic operations, plus a constant number per candidate. For each of the $j_0$ candidates, step 2(a) requires $O(1)$ arithmetic operations. Step 2(b) also requires $O(1)$ arithmetic operations, either due to the use of a factoring oracle, or else, because we can put an arbitrary fixed bound on the amount of effort invested in factoring any given integer. At minimum, this will succeed when the integer in question is prime, which is sufficient for the estimates of Proposition 6.2.11. Step 2(c) requires $O(\text{polylog}(n))$ operations by Proposition 3.2.8. Finally, step 3 requires $O(\ell_{j_0})$ arithmetic operations by Proposition 6.1.11. So the total expected number of arithmetic operations is

$$O(\log(1/M)) + j_0 \cdot O(\text{polylog}(n)) + O(\ell_{j_0}). \tag{6.13}$$

Recall that the $\varepsilon$-region $\mathcal{R}_\varepsilon$, shown in Figure 6.1, contains a disk of radius $\varepsilon^2/4$; therefore, $\text{area}(\mathcal{R}_\varepsilon) \geqslant \frac{\pi}{16}\varepsilon^4$. On the other hand, the square $[-1,1] \times [-1,1]$ is a (not very tight) bounding box for $\mathcal{R}_\varepsilon$. It follows that

$$M = \text{up}(\mathcal{R}_\varepsilon) = \frac{\text{area}(\mathcal{R}_\varepsilon)}{\text{area}(\text{BBox}(\mathcal{R}_\varepsilon))} = \Omega(\varepsilon^4),$$

hence $\log(1/M) = O(\log(1/\varepsilon))$. From (6.11), the expected value of $j_0$ is $O(\log(1/\varepsilon))$. Combining (6.7) with (6.10), we therefore have

$$\ell_{j_0} \leqslant \ell_3 + 2(1 + \log_2 j_0) = O(\log(1/\varepsilon)) + O(\log(\log(1/\varepsilon))) = O(\log(1/\varepsilon)).$$

Now note that for any $i$, $n_i \leqslant 5^{\ell_i+1}$. This, together with the fact that candidates are enumerated in order of increasing denominator exponent, we have $n \leqslant 4^{\ell_{j_0}}$, hence

$$\text{polylog}(n) = O(\text{poly}(\ell_{j_0})) = O(\text{polylog}(1/\varepsilon)).$$

Combining all of these estimates with (6.13), the expected number of arithmetic operations for the algorithm is $O(\text{polylog}(1/\varepsilon))$. Moreover, each individual arithmetic operation can be performed with precision $O(\log(1/\varepsilon))$, taking time $O(\text{polylog}(1/\varepsilon))$. Therefore the total expected time complexity of the algorithm is $O(\text{polylog}(1/\varepsilon))$, as desired. □

## 6.3 Approximate synthesis of special unitaries

The algorithm of the previous section allows us to approximate $z$-rotation up to arbitrarily small accuracy. This method can be used to solve the problem of approximate synthesis of arbitrary special unitaries over the Clifford+$V$ gate set.

**Problem 6.3.1.** Given a special unitary $U$ and a precision $\varepsilon > 0$, construct a Clifford+$V$ circuit $U$ whose $V$-count is as small as possible and such that $\|U - R_z(\theta)\| \leqslant \varepsilon$.

Indeed, an element $U \in \text{SU}(2)$ can always be decomposed as a product of three rotations using *Euler angles.* Hence

$$U = R_z(\theta_1)R_x(\theta_2)R_z(\theta_3).$$

Using the Hadamard gate, the central $x$-rotation can be expressed as a $z$-rotation. Thus

$$U = R_z(\theta_1)HR_z(\theta_2)HR_z(\theta_3).$$

We can therefore use Algorithm 6.2.5 to find a Clifford+$V$ circuit approximating each of the $R_z(\theta_i)$ up to $\varepsilon/3$. Since the Hadamard gate is a Clifford operator, this yields a Clifford+$V$ approximation of $U$ up to $\varepsilon$.

We note that optimality is lost in the process of decomposing an operator as a product of $z$-rotations. Indeed, if we write a special unitary $U$ as

$$U = R_z(\theta_1) H R_z(\theta_2) H R_z(\theta_3)$$

and approximate each $z$-rotation using Algorithm 6.2.5, we obtain a circuit whose length exceeds the optimal one by a factor of 3 in the typical case.

# Chapter 7

## Clifford+$T$ approximate synthesis

In this chapter, we introduce an efficient algorithm to solve the problem of approximate synthesis of special unitaries over the Clifford+$T$ gate set. Recall from Chapter 1 that the $T$ gate is the following matrix

$$T = \begin{bmatrix} 1 & 0 \\ 0 & \omega \end{bmatrix}.$$

The Clifford+$T$ gate set is obtained by adding the $T$ gate to the generators $\omega$, $H$, and $S$ of the Clifford group.

The results presented in this chapter are obtained by adapting the methods of Chapter 6 to the Clifford+$T$ setting. Like in Chapter 6, we first consider the exact synthesis of Clifford+$T$ operators. This provides a characterization of Clifford+$T$ circuits which we then use to define an algorithm for the approximate synthesis of $z$-rotations. Because of the similarities between this chapter and the previous one, we omit most proofs in order to avoid redundancy. However, we explain the differences when they occur.

The approximate synthesis algorithms introduced in this chapter (Algorithm 7.2.5 and Algorithm 7.3.9) have been implemented in Haskell. The implementations are freely available [57].

### 7.1 Exact synthesis of Clifford+$T$ operators

**Problem 7.1.1** (Exact synthesis of Clifford+$T$ operators)**.** Given a unitary $U \in \mathrm{U}(2)$, determine whether there exists a Clifford+$T$ circuit $W$ such that $U = W$ and, in case such a circuit exists, construct one whose $T$-count is minimal.

Problem 7.1.1 was first solved in [41]. A version of Problem 7.1.1 generalized to multi-qubit circuits was solved in [21].

To characterize Clifford+$T$ operators, we consider the following set of unitaries.

**Definition 7.1.2.** The set $\mathcal{T}$ consists of unitary matrices of the form

$$U = \frac{1}{\sqrt{2}^k}\begin{bmatrix}\alpha & \gamma \\ \beta & \delta\end{bmatrix} \tag{7.1}$$

where $k \in \mathbb{N}$ and $\alpha, \beta, \gamma, \delta \in \mathbb{Z}[\omega]$.

We note that $\mathcal{T}$ is the subgroup of $U(2)$ consisting of matrices with entries in $\mathbb{Z}[1/\sqrt{2}, i]$. This is slightly different than the situation in the previous chapter, where we had defined $\mathcal{V}$ to be a strict subset of the group of unitary matrices over $\mathbb{Z}[1/\sqrt{5}, i]$. We will also use a notion of denominator exponent for the elements of $\mathcal{T}$.

**Definition 7.1.3.** Let $U \in \mathcal{T}$ be as in (7.1). The integer $k$ is called a *denominator exponent* of $U$. The least $k$ such that $U$ can be written as in (7.1) is the *least denominator exponent* of $U$. These notions extend naturally to vectors and scalars of the form

$$\frac{1}{\sqrt{2}^k}\begin{bmatrix}\alpha \\ \beta\end{bmatrix} \quad \text{and} \quad \frac{1}{\sqrt{2}^k}\,\alpha, \tag{7.2}$$

where $k \in \mathbb{N}$ and $\alpha, \beta \in \mathbb{Z}[\omega]$.

In the previous chapter, we used the set $\mathcal{V}$ to characterize Clifford+$V$ operators. Similarly, one can prove that Clifford+$T$ operators are exactly the elements of $\mathcal{T}$.

**Proposition 7.1.4** (Kliuchnikov, Maslov, Mosca [42])**.** *A unitary operator $U \in U(2)$ is exactly representable by a Clifford+$T$ circuit if and only if $U \in \mathcal{T}$. Moreover, there exists an efficient algorithm that computes a Clifford+$T$ circuit for $U$ with minimal $T$-count.*

The above proposition can be proved by a technique similar to the one used to establish Proposition 6.1.11. To this end one first shows that every vector of the form (7.2) can be reduced to $e_1 = \left[\begin{smallmatrix}1 \\ 0\end{smallmatrix}\right]$ by applying well-chosen Clifford+$T$ operators. Applying this method to the first column of an element $U$ of $\mathcal{T}$ then yields a circuit for $U$.

Recall that in Proposition 6.1.11, the minimal $V$-count of the operator $U$ was equal to its least $\sqrt{5}$-denominator exponent. The relation between denominator exponent and minimal $T$-count is slightly more complicated.

**Proposition 7.1.5.** *Let $U \in \mathcal{T}$ with least denominator exponent $k$ and minimal $T$-count $t$. Then $2k - 3 \leqslant t \leqslant 2k + 1$.*

*Proof.* See, e.g., [22]. □

## 7.2 Approximate synthesis of $z$-rotations

As in Chapter 6, we consider the problem of approximate synthesis of $z$-rotations.

**Problem 7.2.1.** Given an angle $\theta$ and a precision $\varepsilon > 0$, construct a Clifford+$T$ circuit $U$ whose $T$-count is as small as possible and such that

$$\|U - R_z(\theta)\| \leqslant \varepsilon. \tag{7.3}$$

An algorithm to solve Problem 7.2.1 can be used to solve the problem of approximate synthesis of arbitrary special unitaries using Euler angles, as in Section 6.3.

Our algorithm solving Problem 7.2.1 relies on a reduction of the problem to a grid problem, a Diophantine equation and an exact synthesis problem. This is analogous to the reduction described in the Clifford+$V$ case by Proposition 6.2.4. In the Clifford+$T$ context, we must first show that enumerating candidate solutions in order of increasing denominator exponents allows us to also enumerate candidate solutions in order of minimal $T$-count. This is not immediate, due to Proposition 7.1.5.

**Lemma 7.2.2.** *If $\varepsilon < |1 - e^{i\pi/8}|$, then all solutions to Problem 7.2.1 have the form*

$$U = \frac{1}{\sqrt{2}^k} \begin{bmatrix} u & -t^\dagger \\ t & u^\dagger \end{bmatrix}. \tag{7.4}$$

*If $\varepsilon \geqslant |1 - e^{i\pi/8}|$, then there exists a solution of $T$-count 0 (i.e., a Clifford operator), and it is also of the form* (7.4).

*Proof.* Analogous to the proof of Lemma 6.2.2. □

**Lemma 7.2.3.** *Let $U$ be a unitary operator as in* (7.4) *with least denominator exponent $k$. Then the $T$-count of $U$ is either $2k - 2$ or $2k$. Moreover, if $k > 0$ and $U$ has $T$-count $2k$, then $U' = TUT^\dagger$ has $T$-count $2k - 2$. We further note that $\|R_z(\theta) - U'\| = \|R_z(\theta) - U\|$, so for the purpose of solving* (7.3)*, it does not matter whether $U$ or $U'$ is used. Hence, without loss of generality, we may assume that $U$ as in* (7.4) *always has $T$-count exactly $2k - 2$ when $k > 0$, and* 0 *when $k = 0$.*

*Proof.* The claims about the $T$-counts of $U$ and $U'$ follow by inspection of Figure 2 of [22]. Using the terminology of Definitions 7.4 and 7.6 of [22], this figure shows every possible $k$-residue of a Clifford+$T$ operator, modulo a right action of the group $\langle S, X, \omega \rangle$. Because $U$ is of the form (7.4), only a subset of the $k$-residues is actually possible, and the figure shows that for this subset, the $T$-count is $2k$ or $2k-2$. Moreover, in each of the possible cases where $k > 0$ and $U$ has $T$-count $2k$, the figure also shows that $U' = TUT^\dagger$ has $T$-count $2k-2$.

For the final claim, we have $\|R_z(\theta) - U\| = \|TR_z(\theta)T^\dagger - TUT^\dagger\| = \|R_z(\theta) - U'\|$ because $R_z(\theta)$ and $T$ commute. □

We can now state a reduction for Problem 7.2.1 as we did in Proposition 7.2.4 for Problem 6.2.1.

**Proposition 7.2.4.** *Problem 7.2.1 reduces to a grid problem, a Diophantine equation, and an exact synthesis problem, namely:*

1. *find $k \in \mathbb{N}$ and $\alpha \in \mathbb{Z}[\omega]$ such that $\alpha \in \sqrt{2}^k \mathcal{R}_\varepsilon$ and $\alpha^\bullet \in (-\sqrt{2})^k \overline{\mathcal{D}}$,*

2. *find $\beta \in \mathbb{Z}[\sqrt{2}]$ such that $\beta^\dagger \beta = 2^k - \alpha^\dagger \alpha$, and*

3. *define the unitary matrix $U$ as*

$$U = \frac{1}{\sqrt{2}^k} \begin{bmatrix} \alpha & -\beta^\dagger \\ \beta & \alpha^\dagger \end{bmatrix}$$

*and find a Clifford+$T$ circuit for $U$ or $TUT^\dagger$, whichever has the smaller $T$-count.*

*Moreover, the least $k$ for which the above three problems can be solved yields an optimal solution to Problem 7.2.1.*

Note that item 1 of Proposition 7.2.4 is a grid problem over $\mathbb{Z}[\omega]$. This is in contrast with the corresponding item of Proposition 7.2.4, which was a grid problem over $\mathbb{Z}[i]$. In both cases, we look for points in a scaled $\varepsilon$-region in order of increasing denominator exponent. However, in the Clifford+$T$ case, the desired points must be elements of $\mathbb{Z}[\omega]$. Since $\mathbb{Z}[\omega]$ is dense in $\mathbb{R}^2$, there are infinitely many elements in $\mathcal{R}_\varepsilon \cap \mathbb{Z}[\omega]$ for any fixed denominator exponent. To circumvent this issue, we only consider those elements of $\mathcal{R}_\varepsilon \cap \mathbb{Z}[\omega]$ for which the Diophantine equation of item 2 can potentially be solved. Since we have

$$\alpha^\dagger\alpha + \beta^\dagger\beta = 2^k \implies \alpha \in \sqrt{2}^k\overline{\mathcal{D}} \text{ and } \alpha^\bullet \in (-\sqrt{2})^k\overline{\mathcal{D}},$$

where $\overline{\mathcal{D}}$ is the closed unit disk, the points of interest are precisely the solutions to the scaled grid problem over $\mathbb{Z}[\omega]$ for $\mathcal{R}_\varepsilon$ and $\overline{\mathcal{D}}$.

**Algorithm 7.2.5.** Let $\theta$ and $\varepsilon > 0$ be given.

1. Use the algorithm from Proposition 5.2.37 of Chapter 5 to enumerate the infinite sequence of solutions to the scaled grid problem over $\mathbb{Z}[\omega]$ for $\mathcal{R}_\varepsilon$ and $\overline{\mathcal{D}}$ and $k$ in order of increasing $k$.

2. For each solution $\alpha$:

   (a) Let $\xi = 2^k - \alpha^\dagger\alpha$ and $n = \xi^\bullet\xi$.

   (b) Attempt to find a prime factorization of $n$. If $n \neq 0$ but no prime factorization is found, skip step 2(c) and continue with the next $\alpha$.

   (c) Use the algorithm from Proposition 3.2.9 of Chapter 3 to solve the equation $\beta^\dagger\beta = n$. If a solution $\beta$ exists, go to step 3; otherwise, continue with the next $\alpha$.

3. Define $U$ as
$$U = \frac{1}{\sqrt{2}^k}\begin{bmatrix} \alpha & -\beta^\dagger \\ \beta & \alpha^\dagger \end{bmatrix}$$
and use the exact synthesis algorithm of Proposition 7.1.4 to find a Clifford+$V$ circuit for $U$ or $TUT^\dagger$, whichever has the smallest $T$-count. Output this circuit and stop.

We now state the properties of Algorithm 7.2.5. In most cases it enjoys the same properties as the Clifford+$V$ algorithm.

**Proposition 7.2.6** (Correctness)**.** *If Algorithm 7.2.5 terminates, then it yields a valid solution to the approximate synthesis problem, i.e., it yields a Clifford+T circuit approximating $R_z(\theta)$ up to $\varepsilon$.*

**Proposition 7.2.7** (Optimality in the presence of a factoring oracle). *In the presence of an oracle for integer factoring, the circuit returned by Algorithm 7.2.5 has the smallest T-count of any single-qubit Clifford+T circuit approximating $R_z(\theta)$ up to $\varepsilon$.*

Correctness and optimality are proved like the corresponding propositions in Chapter 6.

Here also, we rely on a number-theoretic assumption on the distribution of primes to establish the remaining properties of the algorithm.

**Hypothesis 7.2.8.** For each $n$ produced in step 2(a) of Algorithm 7.2.5, write $n = 2^j m$, where $m$ is odd. Then $m$ is asymptotically as likely to be prime as a randomly chosen odd number of comparable size. Moreover, the primality of each $m$ can be modelled as an independent random event.

Note that Hypothesis 7.2.8 is slightly different than Hypothesis 6.2.9. Indeed, Hypothesis 6.2.9 makes an additional assumption on the residue class of the integer $m$. Here it is not necessary to make such an assumption, since we can prove that the number $n$ produced in step 2(a) of the algorithm satisfies $n \geqslant 0$ and moreover is such that either $n = 0$ or $n \equiv 1 \pmod 8$. A proof of this fact can be found in Appendix D of [56].

**Proposition 7.2.9** (Near-optimality in the absence of a factoring oracle). *Let $m$ be the T-count of the solution of the approximate synthesis problem found by Algorithm 7.2.5 in the absence of a factoring oracle. Then*

1. *The approximate synthesis problem has at most $O(\log(1/\varepsilon))$ non-equivalent solutions with $T$-count less than $m$.*

2. *The expected value of $m$ is $m'' + O(\log(\log(1/\varepsilon)))$, where $m'$ and $m''$ are the $T$-counts of the optimal and second-to-optimal solutions of the approximate synthesis problem (up to equivalence).*

Note that in Proposition 7.2.9, we use the second-to-optimal solution, rather than the third-to-optimal solution as in Proposition 6.2.11. This is due to the fact that $\sqrt{2} \in \mathbb{Z}[\omega]$ whereas $\sqrt{5} \notin \mathbb{Z}[i]$. Indeed, if $\alpha$ and $\alpha'$ are two solutions of least denominator exponent $k$ and $k'$ with $k \leqslant k'$, then they are both solutions of denominator exponent $k'$. But in the case of the Clifford+$V$ gates, we need to have three solutions to guarantee that two will have the same denominator exponent.

The last property of the algorithm can be proved just like the corresponding one from Chapter 6.

**Proposition 7.2.10.** *Algorithm 7.2.5 runs in expected time* $O(\mathrm{polylog}(1/\varepsilon))$*. This is true whether or not a factorization oracle is used.*

## 7.3 Approximation up to a phase

So far, we have considered the problem of approximate synthesis "on the nose", i.e., the operator $U$ in Problem 7.2.1 was literally required to approximate $R_z(\theta)$ in the operator norm. However, it is well-known that global phases have no observable effect in quantum mechanics, so in quantum computing, it is also common to consider the problem of approximate synthesis "up to a phase". This is made precise in the following definition.

**Problem 7.3.1.** Given $\theta$ and some $\varepsilon > 0$, the *approximate synthesis problem for z-rotations up to a phase* is to find an operator $U$ expressible in the single-qubit Clifford+$T$ gate set, and a unit scalar $\lambda$, such that

$$\|R_z(\theta) - \lambda U\| \leqslant \varepsilon. \tag{7.5}$$

Moreover, it is desirable to find $U$ of smallest possible $T$-count. As before, the norm in (7.5) is the operator norm.

In this section, we will give a version of Algorithm 7.2.5 that optimally solves the approximate synthesis problem up to a phase. The central insight is that it is in fact sufficient to restrict $\lambda$ to only two possible phases, namely $\lambda = 1$ and $\lambda = \sqrt{\omega} = e^{i\pi/8}$.

First, note that if $W$ is a unitary $2 \times 2$-matrix and $\det W = 1$, then $\operatorname{tr} W$ is real. This is obvious, because $\det W = 1$ ensures that the two eigenvalues of $W$ are each other's complex conjugates.

**Lemma 7.3.2.** *Let $W$ be a unitary $2 \times 2$-matrix, and assume that* $\det W = 1$ *and* $\operatorname{tr} W \geqslant 0$*. Then for all unit scalars $\lambda$, we have*

$$\|I - W\| \leqslant \|I - \lambda W\|.$$

*Proof.* We may assume without loss of generality that $W$ is diagonal. Since $\det W = 1$, we can write

$$W = \begin{bmatrix} e^{i\phi} & 0 \\ 0 & e^{-i\phi} \end{bmatrix}$$

for some $\phi$. By symmetry, we can assume without loss of generality that $0 \leqslant \phi \leqslant \pi$. Since $\operatorname{tr} W \geqslant 0$, we have $\phi \leqslant \pi/2$. Now consider a unit scalar $\lambda = e^{i\psi}$, where $-\pi \leqslant \psi \leqslant \pi$. Then $\|I - \lambda W\| = \max\{|1-e^{i(\psi+\phi)}|, |1-e^{i(\psi-\phi)}|\}$ and $\|I-W\| = |1-e^{i\phi}|$. If $\psi \geqslant 0$, then $|1-e^{i\phi}| \leqslant |1-e^{i(\psi+\phi)}|$. Similarly, if $\psi \leqslant 0$, then $|1-e^{i\phi}| \leqslant |1-e^{i(\psi-\phi)}|$. In either case, we have $\|I - W\| \leqslant \|I - \lambda W\|$, as claimed. □

**Lemma 7.3.3.** *Fix $\varepsilon$, a unitary operator $R$ with $\det R = 1$, and a Clifford+$T$ operator $U$. The following are equivalent:*

1. *There exists a unit scalar $\lambda$ such that*

$$\|R - \lambda U\| \leqslant \varepsilon;$$

2. *There exists $n \in \mathbb{Z}$ such that*

$$\|R - e^{in\pi/8} U\| \leqslant \varepsilon.$$

*Proof.* It is obvious that (2) implies (1). For the opposite implication, first note that, because $U$ is a Clifford+$T$ operator, we have $\det U = \omega^k$ for some $k \in \mathbb{Z}$, and therefore $\det(R^{-1}U) = \omega^k$. Let $V = e^{-ik\pi/8} R^{-1} U$, so that $\det V = 1$. If $\operatorname{tr} V \geqslant 0$, let $W = V$; otherwise, let $W = -V$. Either way, we have $W = e^{in\pi/8} R^{-1} U$, where $n \in \mathbb{Z}$, and $\det W = 1$, $\operatorname{tr} W \geqslant 0$. Let $\lambda' = e^{-in\pi/8}\lambda$. By Lemma 7.3.2, we have

$$\begin{array}{rrcl} & \|I - W\| & \leqslant & \|I - \lambda' W\| \\ \Rightarrow & \|I - e^{in\pi/8} R^{-1} U\| & \leqslant & \|I - \lambda' e^{in\pi/8} R^{-1} U\| \\ \Rightarrow & \|R - e^{in\pi/8} U\| & \leqslant & \|R - \lambda' e^{in\pi/8} U\|, \\ \Rightarrow & \|R - e^{in\pi/8} U\| & \leqslant & \|R - \lambda U\|, \end{array}$$

which implies the desired claim. □

*Remark* 7.3.4. A version of Lemma 7.3.3 applies to gate sets other than Clifford+$T$, as long as the gate set has discrete determinants.

**Corollary 7.3.5.** *In Definition 7.3.1, it suffices without loss of generality to consider only the two scalars $\lambda = 1$ and $\lambda = e^{i\pi/8}$.*

*Proof.* Suppose $U$ is a Clifford+$T$ operator satisfying (7.5) for some unit scalar $\lambda$. By Lemma 7.3.3, there exists a $\lambda$ of the form $e^{in\pi/8}$ also satisfying (7.5). Then we can write $\lambda = \omega^k\lambda'$, where $k \in \mathbb{Z}$ and $\lambda' \in \{1, e^{i\pi/8}\}$. Letting $U' = \omega^k U$, we have $\lambda' U' = \lambda U$, and therefore

$$\|R_z(\theta) - \lambda' U'\| \leqslant \varepsilon,$$

as claimed. Moreover, since $\omega = e^{i\pi/4}$ is a Clifford operator, $U$ and $U'$ have the same $T$-count. □

To solve the approximate synthesis problem up to a phase, we therefore need an algorithm for finding optimal solutions of (7.5) in case $\lambda = 1$ and $\lambda = e^{i\pi/8}$. For $\lambda = 1$, this is of course just Algorithm 7.2.5. So all that remains to do is to find an algorithm for solving

$$\|R_z(\theta) - e^{i\pi/8}U\| \leqslant \varepsilon. \tag{7.6}$$

We use a sequence of steps very similar to those of Proposition 7.2.4 to reduce this to a grid problem and a Diophantine equation. We first consider the form of $U$.

**Lemma 7.3.6.** *If $\varepsilon < |1 - e^{i\pi/8}|$, then all solutions of (7.6) have the form*

$$U = \begin{bmatrix} u & -t^\dagger\omega^{-1} \\ t & u^\dagger\omega^{-1} \end{bmatrix}. \tag{7.7}$$

*Proof.* This is completely analogous to the proof of Lemma 7.2.2, using $e^{i\pi/8}U$ in place of $U$. □

Recall that $\delta = 1 + \omega$, and note that $\frac{\delta}{|\delta|} = e^{i\pi/8}$. Also note that $\delta\omega^{-1} = \delta^\dagger$, and that $\delta^{-1} = (\omega - i)/\sqrt{2}$. Suppose that $U$ is of the form (7.7). Let $u' = \delta u$ and $t' = \delta t$. We have:

$$\begin{aligned}
\|R_z(\theta) - e^{i\pi/8}U\| &= \left\| R_z(\theta) - \frac{\delta}{|\delta|}\begin{bmatrix} u & -t^\dagger\omega^{-1} \\ t & u^\dagger\omega^{-1} \end{bmatrix} \right\| \\
&= \left\| R_z(\theta) - \frac{1}{|\delta|}\begin{bmatrix} \delta u & -\delta^\dagger t^\dagger \\ \delta t & \delta^\dagger u^\dagger \end{bmatrix} \right\| \\
&= \left\| R_z(\theta) - \frac{1}{|\delta|}\begin{bmatrix} u' & -t'^\dagger \\ t' & u'^\dagger \end{bmatrix} \right\|.
\end{aligned}$$

Using exactly the same argument as in Proposition 7.2.4, it follows that (7.6) holds if and only if $\frac{u'}{|\delta|} \in \mathcal{R}_\varepsilon$, i.e., $u' \in |\delta|\mathcal{R}_\varepsilon$.

As before, in order for $U$ to be unitary, of course it must satisfy $u^\dagger u + t^\dagger t = 1$, and a necessary condition for this is $u, u^\bullet \in \overline{\mathcal{D}}$. The latter condition can be equivalently re-expressed in terms of $u'$ by requiring $u' \in |\delta|\,\overline{\mathcal{D}}$ and $u'^\bullet \in |\delta^\bullet|\,\overline{\mathcal{D}}$. Therefore, finding solutions to (7.6) of the form (7.7) reduces to the two-dimensional grid problem $u' \in |\delta|\mathcal{R}_\varepsilon$ and $u'^\bullet \in |\delta^\bullet|\,\overline{\mathcal{D}}$, together with the usual Diophantine equation $u^\dagger u + t^\dagger t = 1$. The last remaining piece of the puzzle is to compute the $T$-count of $U$, and in particular, to ensure that potential solutions are found in order of increasing $T$-count.

**Lemma 7.3.7.** *Let $U$ be a Clifford+$T$ operator of the form* (7.7)*, and let $k$ be the least denominator exponent of $u' = \delta u$. Then the $T$-count of $U$ is either $2k-1$ or $2k+1$. Moreover, if $k > 0$ and $U$ has $T$-count $2k+1$, then $U' = TUT^\dagger$ has $T$-count $2k-1$.*

*Proof.* This can be proved by a tedious but easy induction, analogous to Lemma 7.2.3. □

We therefore arrive at the following algorithm for solving (7.6). Here we assume $\varepsilon < |1 - e^{i\pi/8}|$, so that Lemma 7.3.6 applies.

**Algorithm 7.3.8.** Given $\theta$ and $\varepsilon$, let $A = |\delta|\mathcal{R}_\varepsilon$, and let $B = |\delta^\bullet|\,\overline{\mathcal{D}}$.

1. Use Proposition 5.2.37 to enumerate the infinite sequence of solutions to the scaled grid problem over $\mathbb{Z}[\omega]$ for $A$, $B$, and $k$ in order of increasing $k$.

2. For each such solution $u'$:

    (a) Let $\xi = 2^k - u'^\dagger u'$, and $n = \xi^\bullet \xi$.

    (b) Attempt to find a prime factorization of $n$. If $n \neq 0$ but no prime factorization is found, skip step 2(c) and continue with the next $u'$.

    (c) Use the algorithm of Proposition 3.2.9 to solve the equation $t^\dagger t = \xi$. If a solution $t$ exists, go to step 3; otherwise, continue with the next $u'$.

3. Define $U$ as in equation (7.7), let $U' = TUT^\dagger$, and use the exact synthesis algorithm of [42] to find a Clifford+$T$ circuit implementing either $U$ or $U'$, whichever has smaller $T$-count. Output this circuit and stop.

Algorithm 7.3.8 is optimal in the presence of a factoring oracle, and near-optimal in the absence of a factoring oracle, in the same sense as Algorithm 7.2.5. Its expected time complexity is $O(\text{polylog}(1/\varepsilon))$. The proofs are completely analogous to those of the previous section. We then arrive at the following composite algorithm for the approximate synthesis problem for $z$-rotations up to a phase:

**Algorithm 7.3.9** (Approximate synthesis up to a phase)**.** Given $\theta$ and $\varepsilon$, run both Algorithms 7.2.5 and 7.3.8, and return whichever circuit has the smaller $T$-count.

**Proposition 7.3.10** (Correctness, time complexity, and optimality)**.** *Algorithm 7.3.9 yields a valid solution to the approximate synthesis problem up to a phase. It runs in expected time $O(\text{polylog}(1/\varepsilon)$. In the presence of a factoring oracle, the algorithm is optimal, i.e., the returned circuit has the smallest $T$-count of any single-qubit Clifford+$T$ circuit approximating $R_z(\theta)$ up to $\varepsilon$ and up to a phase. Moreover, in the absence of a factoring oracle, the algorithm is near-optimal in the following sense. Let $m$ be the $T$-count of the solution found. Then:*

1. *The approximate synthesis problem up to a phase has an expected number of at most $O(\log(1/\varepsilon))$ non-equivalent solutions with $T$-count less than $m$.*

2. *The expected value of $m$ is $m''' + O(\log(\log(1/\varepsilon)))$, where $m'''$ is the $T$-count of the third-to-optimal solution (up to equivalence) of the approximate synthesis problem up to a phase.*

*Proof.* The correctness and time complexity of Algorithm 7.3.9 follows from that of Algorithms 7.2.5 and 7.3.8. The optimality results follow from those of Algorithms 7.2.5 and 7.3.8, keeping in mind that Algorithm 7.2.5 finds an optimal (or near-optimal) solution for the phase $\lambda = 1$, Algorithm 7.3.8 finds an optimal (or near-optimal) solution for the phase $\lambda = e^{i\pi/8}$, and by Corollary 7.3.5, these are the only two phases that need to be considered.

The only subtlety that must be pointed out is that in part (b) of the near-optimality, we use the $T$-count of the *third*-to-optimal solution, rather than the second-to-optimal one as in Proposition 7.2.9. This is because the optimal and second-to-optimal solutions may belong to Algorithms 7.2.5 and 7.3.8, respectively, so that it may not be until the third-to-optimal solution that the near-optimality result of either Algorithm 7.2.5 or Algorithm 7.3.8 can be invoked. □

*Remark* 7.3.11. Algorithms 7.2.5 and 7.3.8 share the same $\varepsilon$-region up to scaling, and therefore the uprightness computation only needs to be done once.

*Remark* 7.3.12. By Lemmas 7.2.3 and 7.3.7, Algorithm 7.2.5 always produces circuits with even $T$-count, and Algorithm 7.3.8 always produces circuits with odd $T$-count. Instead of running both algorithms to completion, it is possible to interleave the two algorithms, so that all potential solutions are considered in order of increasing $T$-count. This is a slight optimization which does not, however, affect the asymptotic time complexity.

# Chapter 8

# The Proto-Quipper language

In this chapter, we introduce the syntax and operational semantics of the Proto-Quipper language.

## 8.1 From the quantum lambda calculus to Proto-Quipper

Proto-Quipper is based on the quantum lambda calculus. As was discussed in Section 4.4, the execution of programs is modelled in the quantum lambda calculus by a reduction relation defined on closures, which are triples $[Q, L, a]$ consisting of a quantum state $Q$, a list of term variables $L$, and a term $a$. The quantum state is held in a quantum device capable of performing certain operations (applying unitaries, measuring qubits,...). The reduction relation in the quantum lambda calculus is then defined as a probabilistic rewrite procedure on these closures. Typically, the reduction will be classical until a redex involving a quantum constant is reached. At this point, the quantum device will be instructed to perform the appropriate quantum operation. For example: "Apply a Hadamard gate to qubit number 3".

Our approach in designing the Proto-Quipper language was to start with a limited (but still expressive) fragment of the Quipper language and make it completely type-safe. The central aspect of Quipper that we chose to focus on is Quipper's circuit description abilities: to generate and act on quantum circuits. Indeed, Quipper provides the ability to treat circuits as data, and to manipulate them as a whole. For example, Quipper has operators for reversing circuits, decomposing them into gate sets, etc. This is in contrast with the quantum lambda calculus, where one only manipulates qubits and all quantum operations are immediately carried out on a quantum device, not stored for symbolic manipulation.

We therefore extend the quantum lambda calculus with the minimal set of features that makes it Quipper-like. The current version of Proto-Quipper is designed to:

- incorporate Quipper's ability to generate and act on quantum circuits, and to

- provide a linear type system to guarantee that the produced circuits are physically meaningful (in particular, properties like no-cloning are respected).

To achieve these goals, we define Proto-Quipper as a typed lambda calculus, whose type system is similar to that of the quantum lambda calculus. The main difference between Proto-Quipper and the quantum lambda calculus is that the reduction relation of Proto-Quipper is defined on closures $[C, a]$ that consist of a term $a$ and a *circuit state* $C$. Here, the state $C$ represents the circuit currently being built. Instead of having a quantum device capable of performing quantum operations, we assume that we have a *circuit constructor* capable or performing certain circuit building operations (such as appending gates, reversing, etc.). The reduction is then defined as a rewrite procedure on closures. As in the quantum lambda calculus, some redexes will affect the state by sending instructions to the circuit constructor. For example: "Append a Hadamard gate to wire number 3". In the current version of Proto-Quipper, we make the simplifying assumption that no measurements are available, so that the reduction relation is non-probabilistic.

## 8.2 The syntax of Proto-Quipper

In this section, we present in detail the syntax and type system of Proto-Quipper.

**Definition 8.2.1.** The *types* of Proto-Quipper are defined by

$$A, B \quad ::= \quad \mathbf{qubit} \ \big|\ 1 \ \big|\ \mathbf{bool} \ \big|\ A \otimes B \ \big|\ A \multimap B \ \big|\ !A \ \big|\ \mathrm{Circ}(T, U).$$

Among the types, we single out the subset of *quantum data types*

$$T, U \quad ::= \quad \mathbf{qubit} \ \big|\ 1 \ \big|\ T \otimes U.$$

The types 1, **bool**, $A \otimes B$, $A \multimap B$, and $!A$ are inherited from the quantum lambda calculus and should be interpreted as they were in Section 4.4. The elements of **qubit** are references to a logical qubit within a computation. They can be thought of as references to quantum bits on some physical device, or simply as references to quantum wires within the circuit currently being constructed. Elements of quantum data types describe sets of circuit endpoints, and consist of tuples of wire identifiers. We can think of these as describing circuit interfaces. Finally, the type $\mathrm{Circ}(T, U)$ is the set of all circuits having an input interface of type $T$ and an output interface of type $U$.

**Definition 8.2.2.** The *terms* of Proto-Quipper are defined by

$$\begin{array}{rcl} a,b,c & ::= & x \mid q \mid (t,C,a) \mid \texttt{True} \mid \texttt{False} \mid \langle a,b\rangle \mid * \mid ab \mid \lambda x.a \mid \\ & & \mathit{rev} \mid \mathit{unbox} \mid \mathit{box}^T \mid \texttt{if}\ a\ \texttt{then}\ b\ \texttt{else}\ c \mid \texttt{let}\ * = a\ \texttt{in}\ b \mid \\ & & \texttt{let}\ \langle x,y\rangle = a\ \texttt{in}\ b. \end{array}$$

where $x$ and $y$ come from a countable set $\mathcal{V}$ of *term variables*, $q$ comes from a countable set $\mathcal{Q}$ of *quantum variables*, and $C$ comes from a countable set $\mathcal{C}$ of *circuit constants.* Among the terms, we single out the subset of *quantum data terms*

$$t,u \quad ::= \quad q \mid * \mid \langle t,u\rangle.$$

Moreover, we assume that $\mathcal{C}$ is equipped with two functions $\mathrm{In},\mathrm{Out}\colon \mathcal{C} \to \mathcal{P}_f(\mathcal{Q})$ and that $\mathcal{Q}$ is well-ordered. Here, $\mathcal{P}_f(\mathcal{Q})$ denotes the set of finite subsets of $\mathcal{Q}$.

The meaning of most terms is intended to be the standard one. For example $\langle a,b\rangle$ is the pair of $a$ and $b$, `True` and `False` are the booleans and $\lambda x.a$ is the function which maps $x$ to $a$. We briefly discuss the meaning of the more unusual terms.

- A circuit constant $C$ represents a low-level quantum circuit. Because it would be complicated, and somewhat besides the point, to define a formal language for describing low-level quantum circuits, Proto-Quipper assumes that there exists a constant symbol for *every* possible quantum circuit. Each circuit $C$ is equipped with a finite set of *inputs* and a finite set of *outputs*, which are subsets of the set of quantum variables $\mathcal{Q}$. Proto-Quipper's abstract treatment of quantum circuits is further explained in Section 8.3.

- The term $(t,C,a)$ represents a quantum circuit, regarded as Proto-Quipper data. The purpose of the terms $t$ and $a$ is to provide structure on the (otherwise unordered) sets of inputs and outputs of $C$, so that these inputs and outputs can take the shape of Proto-Quipper quantum data. For example, suppose that $C$ is a circuit with inputs $\{q_1,q_2,q_3\}$ and outputs $\{q_4,q_5,q_6\}$. Then the term

$$(\langle q_2,\langle q_3,q_1\rangle\rangle, C, \langle\langle q_4,q_6\rangle,q_5\rangle)$$

represents the circuit $C$, but also specifies what it means to apply this circuit to a quantum data term $\langle p, \langle r, s\rangle\rangle$. Namely, in this case, the circuit inputs $q_2$, $q_3$, and $q_1$ will be applied to qubits $p$, $r$, and $s$, respectively. Moreover, if the output of this circuit is to be matched against the pattern $\langle\langle x, y\rangle, z\rangle$, then the variables $x$, $y$, and $z$ will be bound, respectively, to the quantum bits at endpoints $q_4$, $q_6$, and $q_5$.

Terms of the form $(t, C, a)$ are not intended to be written by the user of the programming language; in fact, a Proto-Quipper implementation would not provide a concrete syntax for such terms. Rather, these terms are internally generated during the *evaluation* of Proto-Quipper programs. However, the circuits for certain basic gates may be made available to the user as pre-defined symbols.

- $box^T$ is a built-in function to turn a circuit-producing function (for example, a function of type $T \multimap U$) into a circuit regarded as data (for example, of type $\mathrm{Circ}(T, U)$).

- *unbox* is a built-in function for turning a circuit regarded as data into a circuit-producing function. It is an inverse of $box^T$.

- *rev* is a built-in function for reversing a low-level circuit.

Note that the term $box^T$ is parameterized by a type $T$. This *Church-style* typing of the language is the reason why types were introduced before terms. Also note that in a term like $(t, C, a)$, $t$ is assumed to be a quantum data term, but $a$ is not. The type system to be introduced below will guarantee that even though $a$ is not yet a quantum data term it will eventually reduce to one.

*Examples* 8.2.3. Suppose that $H$ is the circuit constant for the Hadamard gate. The term $\langle q_1, H, q_2\rangle$ then represents the circuit consisting only of the Hadamard gate, regarded as Proto-Quipper data. We can then define the circuit producing function $H = unbox\langle q_1, H, q_2\rangle$. Similarly, if $CNOT$ is the circuit constant for the controlled-not gate, then the term $CNOT = unbox\langle\langle q_1, q_2\rangle, CNOT, \langle q_3, q_4\rangle\rangle$ is the corresponding circuit producing function. In an implementation of the Proto-Quipper language, a finite set of such circuit producing functions would be provided as basic operations. For example, a candidate such gate set would consist of the Clifford+$T$ gate set extended with the controlled-not gate: $H, S, T, CNOT$. Basic gates can then be combined. For example, the term

$$\lambda x.\, T(S(Hx))$$

is the circuit producing function which applies in sequence the $H$, $S$, and $T$ gates. Using the $box^T$ operator, we can turn this circuit producing function into a circuit

$$box^{\mathbf{qubit}}(\lambda x.\, T(S(Hx))).$$

As in the quantum lambda calculus, the operational semantics of Proto-Quipper will be defined according to a call-by-value reduction strategy. We therefore define what it means, for a term of Proto-Quipper, to be a value.

**Definition 8.2.4.** The *values* of Proto-Quipper are defined by

$$\begin{array}{rcl} v, w & ::= & x \mid q \mid (t, C, u) \mid \texttt{True} \mid \texttt{False} \mid \langle v, w\rangle \mid \\ & & * \mid \lambda x.a \mid box^T \mid rev \mid unbox \mid unbox\ v. \end{array}$$

Note that according to Definition 8.2.4, some applications are values, namely terms of the form $unbox\ v$. This is consistent with the meaning of the $unbox$ constant discussed above. Indeed, if $unbox$ turns a circuit into a circuit-generating function, then a term of the form $unbox\ v$ should be seen as a function awaiting an argument, much like a term of the form $\lambda x.a$, and therefore considered a value.

We now introduce some useful syntactic operations on types and terms. We start by defining the notion of free variable for Proto-Quipper terms.

**Definition 8.2.5.** The set of *free (term) variables* of a term $a$, written $\mathrm{FV}(a)$, is defined as

- $\mathrm{FV}(x) = \{x\}$,
- $\mathrm{FV}(\langle a, b\rangle) = \mathrm{FV}(a) \cup \mathrm{FV}(b)$,
- $\mathrm{FV}(ab) = \mathrm{FV}(a) \cup \mathrm{FV}(b)$,
- $\mathrm{FV}(\lambda x.a) = \mathrm{FV}(a) \setminus \{x\}$,
- $\mathrm{FV}(\texttt{if}\ a\ \texttt{then}\ b\ \texttt{else}\ c) = \mathrm{FV}(a) \cup \mathrm{FV}(b) \cup \mathrm{FV}(c)$,
- $\mathrm{FV}(\texttt{let}\ * = a\ \texttt{in}\ b) = \mathrm{FV}(a) \cup \mathrm{FV}(b)$,

- $\mathrm{FV}(\texttt{let}\ \langle x,y\rangle = a\ \texttt{in}\ b) = \mathrm{FV}(a) \cup (\mathrm{FV}(b) \setminus \{x,y\})$,
- $\mathrm{FV}((t,C,a)) = \mathrm{FV}(a)$, and
- $\mathrm{FV}(a) = \emptyset$ in all remaining cases.

The above definition of free variables extends the standard one. Note that the free variables of a term of the form $(t,C,a)$ are the free variables of $a$. This is justified since no variables ever appear in the quantum data term $t$.

The notions of $\alpha$-equivalence, capture-avoiding substitution, etc., are defined in a straightforward manner.

By analogy with the free term variables of a term, we introduce a notion of *quantum variable of a term.*

**Definition 8.2.6.** The set of *free quantum variables* of a term $a$, written $\mathrm{FQ}(a)$, is defined as

- $\mathrm{FQ}(q) = \{q\}$,
- $\mathrm{FQ}(\langle a,b\rangle) = \mathrm{FQ}(a) \cup \mathrm{FQ}(b)$,
- $\mathrm{FQ}(ab) = \mathrm{FQ}(a) \cup \mathrm{FQ}(b)$,
- $\mathrm{FQ}(\lambda x.a) = \mathrm{FQ}(a)$,
- $\mathrm{FQ}(\texttt{if}\ a\ \texttt{then}\ b\ \texttt{else}\ c) = \mathrm{FQ}(a) \cup \mathrm{FQ}(b) \cup \mathrm{FQ}(c)$,
- $\mathrm{FQ}(\texttt{let}\ * = a\ \texttt{in}\ b) = \mathrm{FQ}(a) \cup \mathrm{FQ}(b)$,
- $\mathrm{FQ}(\texttt{let}\ \langle x,y\rangle = a\ \texttt{in}\ b) = \mathrm{FQ}(a) \cup \mathrm{FQ}(b)$, and
- $\mathrm{FQ}(a) = \emptyset$ in all remaining cases.

Note that $\mathrm{FQ}((t,C,a)) = \emptyset$. This reflects the idea that the quantum variables appearing in $t$ and $a$ are "bound" in $(t,C,a)$.

To append circuits, we will need to be able to express the way in which wires should be connected. For this, we use the notion of a *binding.*

**Definition 8.2.7.** A *finite bijection* on a set $X$ is a bijection between two finite subsets of $X$. We write $\mathrm{Bij}_f(X)$ for the set of finite bijections on $X$. The domain and codomain of a finite bijection $\mathfrak{b}$ are denoted $\mathrm{dom}(\mathfrak{b})$ and $\mathrm{cod}(\mathfrak{b})$, respectively.

**Definition 8.2.8.** A *binding* is a finite bijection on $\mathcal{Q}$. We will usually denote bindings by $\mathfrak{b}$.

**Definition 8.2.9.** If $a$ is a term, $\mathfrak{b}$ is a binding and $\mathrm{FQ}(a) = \{q_1, \ldots, q_n\} \subseteq \mathrm{dom}(\mathfrak{b})$, then $\mathfrak{b}(a)$ is the following term

$$\mathfrak{b}(a) = a[\mathfrak{b}(q_1)/q_1, \ldots, \mathfrak{b}(q_n)/q_n].$$

**Definition 8.2.10.** The partial function $\mathit{bind} : \mathtt{QDataTerm}^2 \to \mathrm{Bij}_f(\mathcal{Q})$ is defined as

- $\mathit{bind}(*, *) = \emptyset$;
- $\mathit{bind}(q_1, q_2) = \{(q_1, q_2)\}$;
- $\mathit{bind}(\langle t_1, t_2\rangle, \langle u_1, u_2\rangle) = \mathit{bind}(t_1, u_1) \uplus \mathit{bind}(t_2, u_2)$, provided that $\mathit{bind}(t_1, u_1) \cap \mathit{bind}(t_2, u_2) = \emptyset$;
- $\mathit{bind}(t, u) =$ undefined, in all remaining cases.

**Definition 8.2.11.** Let $T$ be a quantum data type and $X$ a finite subset of $\mathcal{Q}$. An $X$-*specimen* for $T$ is quantum data term written $\mathtt{Spec}_X(T)$ defined as

- $\mathtt{Spec}_X(1) = *$,
- $\mathtt{Spec}_X(\mathbf{qubit}) = q$ where $q$ is the smallest quantum index of $\mathcal{Q} \setminus X$,
- $\mathtt{Spec}_X(T \otimes U) = \langle t, u\rangle$ where $t = \mathtt{Spec}_X(T)$ and $u = \mathtt{Spec}_{X \cup \mathrm{FQ}(t)}(U)$.

Informally, an $X$-specimen for $T$ is a quantum data term $t$ that is “fresh” with respect to the quantum variables appearing in $X$. If $X$ is clear from the context, we simply write $\mathtt{Spec}(T)$. Note that the definition of specimen uses the fact that $\mathcal{Q}$ is well-ordered.

As in the quantum lambda calculus, we use a subtyping relation to deal with the ! modality.

**Definition 8.2.12.** The *subtyping relation* $<:$ is the smallest relation on types satisfying the rules given in Figure 8.1.

Note that the subtyping of $A \multimap B$ and $\mathrm{Circ}(A, B)$ is *contravariant* in the left argument, i.e., $A <: A'$ implies $A' \multimap B <: A \multimap B$.

$$\frac{}{\mathbf{qubit} <: \mathbf{qubit}} \qquad \frac{}{1 <: 1} \qquad \frac{}{\mathbf{bool} <: \mathbf{bool}}$$

$$\frac{A_1 <: B_1 \quad A_2 <: B_2}{(A_1 \otimes A_2) <: (B_1 \otimes B_2)} \qquad \frac{A_2 <: A_1 \quad B_1 <: B_2}{(A_1 \multimap B_1) <: (A_2 \multimap B_2)}$$

$$\frac{A_2 <: A_1 \quad B_1 <: B_2}{\mathrm{Circ}(A_1, B_1) <: \mathrm{Circ}(A_2, B_2)}$$

$$\frac{A <: B \quad (n = 0 \Rightarrow m = 0)}{!^n A <: !^m B}$$

Figure 8.1: Subtyping rules for Proto-Quipper.

*Remark* 8.2.13. If $A <: B$ then:

1. if $A \in \{\mathbf{qubit}, 1, \mathbf{bool}\}$, then $A = B$;
2. if $A = A_1 \otimes A_2$, then $B = B_1 \otimes B_2$, $A_1 <: B_1$ and $A_2 <: B_2$;
3. if $A = A_1 \multimap A_2$, then $B = B_1 \multimap B_2$, $B_1 <: A_1$ and $A_2 <: B_2$;
4. if $A = \mathrm{Circ}(A_1, A_2)$, then $B = \mathrm{Circ}(B_1, B_2)$, $B_1 <: A_1$ and $A_2 <: B_2$;
5. if $B = !B'$, then $A = !A'$ and $A' <: B'$;
6. if $A$ is not of the form $!A'$, then $B$ is not of the form $!B'$.

**Proposition 8.2.14.** *The subtyping relation is reflexive and transitive.*

As in the quantum lambda calculus, the following subtyping rule is derivable

$$\frac{}{!A <: A} \ .$$

**Definition 8.2.15.** A *typing context* is a finite set $\{x_1 : A_1, \ldots, x_n : A_n\}$ of pairs of a variable and a type, such that no variable occurs more than once. A *quantum context* is a finite set of quantum variables. The expressions of the form $x : A$ in a typing context are called *type declarations.*

We write $\Gamma$ or $\Delta$ for a typing context and $Q$ for a quantum context. We also adopt the previous notational conventions when dealing with typing contexts: $|\Gamma|$, $\Gamma(x_i)$, $!\Gamma$, and $\Gamma <: \Gamma'$. Moreover, we still write $\Gamma, \Gamma'$ to denote the union of two contexts, which is defined when $|\Gamma| \cap |\Gamma'| = \emptyset$.

**Definition 8.2.16.** Let $T, U$ be quantum data types. For each of the constants $\mathit{box}^T$, *unbox*, and *rev*, we introduce a type as follows

- $A_{\mathit{box}^T}(T, U) = !(T \multimap U) \multimap !\,\mathrm{Circ}(T, U)$,
- $A_{\mathit{unbox}}(T, U) = \mathrm{Circ}(T, U) \multimap !(T \multimap U)$, and
- $A_{\mathit{rev}}(T, U) = \mathrm{Circ}(T, U) \multimap !\,\mathrm{Circ}(U, T)$.

**Definition 8.2.17.** A *typing judgment* is an expression of the form:

$$\Gamma; Q \vdash a : A$$

where $\Gamma$ is a typing context, $Q$ is a quantum context, $a$ is a term and $A$ is a type. A typing judgment is *valid* if it can be inferred from the rules given in Figure 8.2. In the rule $(\mathit{cst})$, $c$ ranges over the set $\{\mathit{box}^T, \mathit{unbox}, \mathit{rev}\}$. Each typing rule carries an implicit side condition that the judgements appearing in it are well-formed. In particular, a rule containing a context of the form $\Gamma_1, \Gamma_2$ may not be applied unless $|\Gamma_1| \cap |\Gamma_2| = \emptyset$.

Note that in the typing judgements of Proto-Quipper, quantum variables and variables are kept separate. As a result, we do not have to specify that $q : \mathbf{qubit}$ for every quantum variable $q$ since the typing rules implicitly enforce this. However, when a future version of Proto-Quipper will be equipped with the ability to manipulate quantum *and* classical wires, the type of a wire might have to be explicitly stated.

As a first illustration of the safety properties of the type system, note that the $(\otimes_i)$ rule ensures that $\lambda x.\langle x, x\rangle$ cannot be given the type $\mathbf{qubit} \multimap \mathbf{qubit} \otimes \mathbf{qubit}$.

## 8.3 The operational semantics of Proto-Quipper

As mentioned Section 8.1, the reduction relation for Proto-Quipper is defined in the presence of a *circuit constructor*. This is a device capable of performing certain basic circuit building operations. It is not necessary to have a detailed description of the inner workings of this device. In fact, all that is required for the definition of Proto-Quipper's operational semantics is the existence of some primitive operations. We now axiomatize these operations. Their intuitive meaning will be explained following Definition 8.3.1.

$$\frac{A <: B}{!\Delta, x : A; \emptyset \vdash x : B}\,(ax_c) \qquad \frac{}{!\Delta; \{q\} \vdash q : \mathbf{qubit}}\,(ax_q)$$

$$\frac{!A_c(T,U) <: B}{!\Delta; \emptyset \vdash c : B}\,(cst) \qquad \frac{}{!\Delta; \emptyset \vdash * : !^n 1}\,(*_i)$$

$$\frac{\Gamma, x : A; Q \vdash b : B}{\Gamma; Q \vdash \lambda x.b : A \multimap B}\,(\lambda_1) \qquad \frac{!\Delta, x : A; \emptyset \vdash b : B}{!\Delta; \emptyset \vdash \lambda x.b :\ !^{n+1}(A \multimap B)}\,(\lambda_2)$$

$$\frac{\Gamma_1, !\Delta; Q_1 \vdash c : A \multimap B \quad \Gamma_2, !\Delta; Q_2 \vdash a : A}{\Gamma_1, \Gamma_2, !\Delta; Q_1, Q_2 \vdash ca : B}\,(app)$$

$$\frac{\Gamma_1, !\Delta; Q_1 \vdash a : !^n A \quad \Gamma_2, !\Delta; Q_2 \vdash b : !^n B}{\Gamma_1, \Gamma_2, !\Delta; Q_1, Q_2 \vdash \langle a, b \rangle : !^n(A \otimes B)}\,(\otimes_i)$$

$$\frac{\Gamma_1, !\Delta; Q_1 \vdash b : !^n(B_1 \otimes B_2) \quad \Gamma_2, !\Delta, x : !^n B_1, y : !^n B_2; Q_2 \vdash a : A}{\Gamma_1, \Gamma_2, !\Delta; Q_1, Q_2 \vdash \mathtt{let}\ \langle x, y \rangle = b\ \mathtt{in}\ a : A}\,(\otimes_e)$$

$$\frac{\Gamma_1, !\Delta; Q_1 \vdash b : !^n 1 \quad \Gamma_2, !\Delta; Q_2 \vdash a : A}{\Gamma_1, \Gamma_2, !\Delta; Q_1, Q_2 \vdash \mathtt{let}\ * = b\ \mathtt{in}\ a : A}\,(*_e)$$

$$\frac{}{!\Delta; \emptyset \vdash \mathtt{True} : !^n\mathbf{bool}}\,(\top) \qquad \frac{}{!\Delta; \emptyset \vdash \mathtt{False} : !^n\mathbf{bool}}\,(\bot)$$

$$\frac{\Gamma_1, !\Delta; Q_1 \vdash b : \mathbf{bool} \quad \Gamma_2, !\Delta; Q_2 \vdash a_1 : A \quad \Gamma_2, !\Delta; Q_2 \vdash a_2 : A}{\Gamma_1, \Gamma_2, !\Delta; Q_1, Q_2 \vdash \mathtt{if}\ b\ \mathtt{then}\ a_1\ \mathtt{else}\ a_2 : A}\,(if)$$

$$\frac{Q_1 \vdash t : T \quad !\Delta; Q_2 \vdash a : U \quad \mathrm{In}(C) = Q_1 \quad \mathrm{Out}(C) = Q_2}{!\Delta; \emptyset \vdash (t, C, a) : !^n\,\mathrm{Circ}(T, U)}\,(circ)$$

Figure 8.2: Typing rules for Proto-Quipper.

**Definition 8.3.1.** A *circuit constructor* consists of a pair of countable sets $\langle Q, S \rangle$ together with the following maps

- $\mathtt{New}\colon \mathscr{P}_f(Q) \to S$,
- $\mathtt{In}\colon S \to \mathscr{P}_f(Q)$,
- $\mathtt{Out}\colon S \to \mathscr{P}_f(Q)$,
- $\mathtt{Rev}\colon S \to S$,
- $\mathtt{Append}\colon S \times S \times \mathrm{Bij}_f(Q) \to S \times \mathrm{Bij}_f(Q)$

satisfying the following conditions

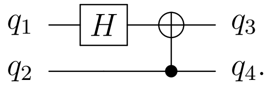
1. $\mathtt{Rev} \circ \mathtt{Rev} = 1_S$,

2. $\mathtt{In} \circ \mathtt{Rev} = \mathtt{Out}$ and $\mathtt{Out} \circ \mathtt{Rev} = \mathtt{In}$,

3. $\mathtt{In} \circ \mathtt{New} = \mathtt{Out} \circ \mathtt{New} = 1_{\mathcal{P}_f(\mathcal{Q})}$, and

4. if $\mathtt{Append}(C, D, b) = (C', b')$ and $\mathrm{dom}(b) \subseteq \mathtt{Out}(C)$ and $\mathrm{cod}(b) = \mathtt{In}(D)$, then
   (a) $\mathtt{In}(C') = \mathtt{In}(C)$,
   (b) $\mathrm{dom}(b') = \mathtt{Out}(D)$ and $\mathrm{cod}(b') \subseteq \mathtt{Out}(C')$ and
   (c) $\mathtt{Out}(C') = (\mathtt{Out}(C) \setminus \mathrm{dom}(b)) \mathbin{\dot\cup} \mathrm{cod}(b')$.

If $\langle Q, S \rangle$ is a circuit constructor, we call the elements of $S$ *circuit states* and the elements of $Q$ *wire identifiers*. We now explain the intended meaning of a circuit constructor and its constituents. An element $C \in S$ is a quantum circuit, such as

$C \quad =$ 

Each circuit has a finite set of inputs and a finite set of outputs, given by the functions In and Out. For example, $\mathtt{In}(C) = \{q_1, q_2\}$ and $\mathtt{Out}(C) = \{q_3, q_4\}$. For $X \subseteq Q$, the circuit $\mathtt{New}(X)$ is the identity circuit with inputs and outputs $X$; for example,

$\mathtt{New}(q_1, q_2, q_3) \quad =$
$q_1$
$q_1$
$q_2$
$q_2$
$q_3$
$q_3$.

The operator Rev reverses a circuit, swapping its inputs and outputs in the process. When $(C', b') = \mathtt{Append}(C, D, b)$, the circuit $C'$ is obtained by appending the circuit $D$ to the end of the circuit $C$. The function $b$ is used to specify along which wires to compose $C$ and $D$ while the function $b'$ updates the wire names post composition. An illustration of this is given in Figure 8.3.

We note that the axiomatization of Definition 8.3.1 does not mention the concept of a *gate*. Indeed, any gate is a circuit, and thus a member of the set $S$; conversely, any circuit can be used as a gate. In Proto-Quipper, we simply assume that certain members of $S$ are available as pre-defined constants, serving as "elementary" gates. The operation of appending a gate to a circuit is subsumed by the more general operation of composing circuits.

Proto-Quipper's quantum variables and circuit constants are supposed to be the syntactic representatives of a circuit constructor's wire identifiers. This idea is formalized in the following definition.

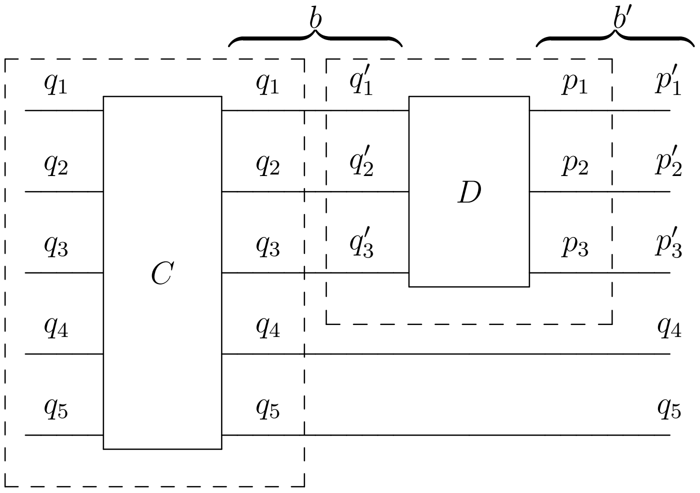


Figure 8.3: A representation of $\texttt{Append}(C, D, b)$.

**Definition 8.3.2.** A circuit constructor $\langle Q, S\rangle$ is *adequate* if it can be equipped with bijections $\texttt{Wire}\colon \mathcal{Q} \to Q$ and $\texttt{Name}\colon \mathcal{C} \to S$ such that:

$$\mathrm{In} = \texttt{Wire}' \circ \texttt{In} \circ \texttt{Name} \quad \text{and} \quad \mathrm{Out} = \texttt{Wire}' \circ \texttt{Out} \circ \texttt{Name}$$

where $\texttt{Wire}'$ denotes the lifting of $\texttt{Wire}^{-1}$ from $\mathcal{Q}$ to $\mathcal{P}(\mathcal{Q})$.

*Remark* 8.3.3. The existence of the bijections $\texttt{Wire}$ and $\texttt{Name}$ has the following consequences:

- $\mathcal{C}$ can be equipped with an involution:

  $$(.)^{-1} = \texttt{Name} \circ \texttt{Rev} \circ \texttt{Name}^{-1}\colon \mathcal{C} \to \mathcal{C}$$

  such that $\mathrm{In}(C^{-1}) = \mathrm{Out}(C)$ and $\mathrm{Out}(C^{-1}) = \mathrm{In}(C)$.

- If $t$ and $u$ are quantum data terms such that $\mathit{bind}(t, u) = \mathfrak{b}$, then we can define $b = \texttt{Wire} \circ \mathfrak{b} \circ \texttt{Wire}^{-1} \in \mathrm{Bij}_f(Q)$.

From now on, we always assume an adequate circuit constructor. Moreover, we work under the simplifying assumptions that $\mathcal{Q} = Q$, $\mathcal{C} = S$, $\texttt{Wire} = 1_Q$, and $\texttt{Name} = 1_S$. This notably implies that $\mathrm{In} = \texttt{In}$ and $\mathrm{Out} = \texttt{Out}$.

We are now in a position to define Proto-Quipper's operational semantics.

**Definition 8.3.4.** Let $\langle Q, S \rangle$ be an adequate circuit constructor. A *closure* is a pair $[C, a]$ where $C \in S$, $a$ is a term and $\mathrm{FQ}(t) \subseteq \mathtt{Out}(C)$.

**Definition 8.3.5.** The *one-step reduction relation*, written $\to$, is defined on closures by the rules given in Figure 8.4. The *reduction relation*, written $\to^*$, is defined to be the reflexive and transitive closure of $\to$.

The rules are separated in three groups. The first group and second group contain the *congruence rules* and *classical rules* respectively. Except for the rule for $(t, D, a)$, these rules are standard. They describe a call-by-value reduction strategy. The circuit generating rule for $\mathit{rev}(t, C, t')$ rule is straightforward. We briefly discuss the remaining rules.

The rule for $\mathit{box}^T(v)$ is to be understood as follows. To reduce a closure of the form $[C, \mathit{box}^T(v)]$, start by generating a specimen of type $T$. Then apply the function $v$ on the input $t$ in the context of an empty circuit of the appropriate arity. By the congruence rule for $(t, D, a)$, this computation will continue until a value is reached, i.e., a term of the form $(t, D, t')$. Note that while this computation is taking place, the state $C$ is not accessible. When a value of the form $(t, D, t')$ is reached, the construction of $C$ can resume. Note that it was necessary to know the type $T$ in order to generate the appropriate specimen. This explains the choice of a Church-style typing of the *box* operator.

The rule for $(\mathit{unbox}(u, D, u'))v$ will first generate a binding from $v$ and the input $u$ of $D$. Then, it will compose $C$ and $D$ along that binding and update the names of the wire identifiers appearing in $u'$ according to $\mathfrak{b}'$.

The recursive nature of the reduction rules explains why closures are not required to satisfy $\mathrm{FQ}(a) = \mathtt{Out}(C)$. The requirement that $\mathrm{FQ}(a) \subseteq \mathtt{Out}(C)$ is justified by the idea that a term should not affect a wire outside of $C$. But if we also asked for the opposite inclusion, it would not be possible to define a recursive reduction in a straightforward way. For example, the reduction of a pair is done component-wise: to reduce $\langle a, b \rangle$ one first reduces $b$. The simplest way to express this in terms of closures is to carry the whole circuit state along. This implies that if both $a$ and $b$ contain wire identifiers, then the equality $\mathrm{FQ}(a) = \mathtt{Out}(C)$ cannot be satisfied.

Unlike in the quantum lambda calculus, Proto-Quipper's reduction is not probabilistic, in the sense that the right member of any reduction rule is a unique closure.

$$\frac{[C,a] \to [C',a']}{[C,ab] \to [C',a'b]}\ (\mathit{fun}) \qquad \frac{[C,b] \to [C',b']}{[C,vb] \to [C',vb']}\ (\mathit{arg})$$

$$\frac{[C,b] \to [C',b']}{[C,\langle a,b\rangle] \to [C',\langle a,b'\rangle]}\ (\mathit{right}) \qquad \frac{[C,a] \to [C',a']}{[C,\langle a,v\rangle] \to [C',\langle a',v\rangle]}\ (\mathit{left})$$

$$\frac{[C,a] \to [C',a']}{[C,\texttt{let}\ * = a\ \texttt{in}\ b] \to [C',\texttt{let}\ * = a'\ \texttt{in}\ b]}\ (\mathit{let}*)$$

$$\frac{[C,a] \to [C',a']}{[C,\texttt{let}\ \langle x,y\rangle = a\ \texttt{in}\ b] \to [C',\texttt{let}\ \langle x,y\rangle = a'\ \texttt{in}\ b]}\ (\mathit{let})$$

$$\frac{[C,a] \to [C',a']}{[C,\texttt{if}\ a\ \texttt{then}\ b\ \texttt{else}\ c] \to [C',\texttt{if}\ a'\ \texttt{then}\ b\ \texttt{else}\ c]}\ (\mathit{cond})$$

$$\frac{[D,a] \to [D',a']}{[C,(t,D,a)] \to [C,(t,D',a')]}\ (\mathit{circ})$$

$$\frac{}{[C,(\lambda x.a)v] \to [C,a[v/x]]}\ (\beta)$$

$$\frac{}{[C,\texttt{let}\ * = *\ \texttt{in}\ a] \to [C,a]}\ (\mathit{unit})$$

$$\frac{}{[C,\texttt{let}\ \langle x,y\rangle = \langle v,w\rangle\ \texttt{in}\ a] \to [C,a[v/x,w/y]]}\ (\mathit{pair})$$

$$\frac{}{[C,\texttt{if False then}\ a\ \texttt{else}\ b] \to [C,b]}\ (\mathit{ifF})$$

$$\frac{}{[C,\texttt{if True then}\ a\ \texttt{else}\ b] \to [C,a]}\ (\mathit{ifT})$$

$$\frac{\texttt{Spec}_{\mathrm{FQ}(v)}(T) = t \quad \texttt{new}(\mathrm{FQ}(t)) = D}{[C,\mathit{box}^T(v)] \to [C,(t,D,vt)]}\ (\mathit{box})$$

$$\frac{\mathit{bind}(v,u) = \mathfrak{b} \quad \texttt{Append}(C,D,\mathfrak{b}) = (C',\mathfrak{b}') \quad \mathrm{FQ}(u') \subseteq \mathrm{dom}(\mathfrak{b}')}{[C,(\mathit{unbox}(u,D,u'))v] \to [C',\mathfrak{b}'(u')]}\ (\mathit{unbox})$$

$$\frac{}{[C,\mathit{rev}(t,C,t')] \to [C,(t',C^{-1},t)]}\ (\mathit{rev})$$

Figure 8.4: Reduction rules for Proto-Quipper.

The following proposition establishes that Proto-Quipper's reduction is moreover *deterministic.*

**Proposition 8.3.6.** *If* $[C, a]$ *is a closure, then at most one reduction rule applies to it.*

*Proof.* By case distinction on $a$. □

To close this chapter, we illustrate the reduction of Proto-Quipper with an example. Assume that we are given the basic circuit generating functions *H*, *S*, and *CNOT* of Examples 8.2.3 and let $F$ be the following term

$$F = \lambda z.(\texttt{let}\ \langle x, y\rangle = z\ \texttt{in}\ \ CNOT\langle Hx, Sy\rangle)$$

Since $F$ can be given the type $\mathbf{qubit} \otimes \mathbf{qubit} \multimap \mathbf{qubit} \otimes \mathbf{qubit}$, we can use $box^{\mathbf{qubit}\otimes\mathbf{qubit}}$ to turn $F$ into a Proto-Quipper circuit. Now consider the closure

$$[-, box^{\mathbf{qubit}\otimes\mathbf{qubit}}\ F] \tag{8.1}$$

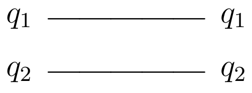
where $-$ is any circuit state. The (*box*) rule applies, so that a specimen of type $\mathbf{qubit} \otimes \mathbf{qubit}$ is created, say $\langle q_1, q_2\rangle$, and (8.1) reduces to

$$[-, (\langle q_1, q_2\rangle, C, F\langle q_1, q_2\rangle)]$$

where $C = \texttt{new}(\{q_1, q_2\})$ is the empty circuit on $\{q_1, q_2\}$. Since $F\langle q_1, q_2\rangle$ is not a value, the (*circ*) rule applies. This means that we consider the closure

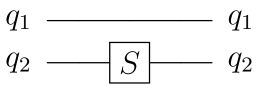
$$[C, F\langle q_1, q_2\rangle].$$

We now repeatedly consider reducts of this closure until a value is reached. For clarity, we represent circuit states as circuits. In two classical reductions we reach the closure



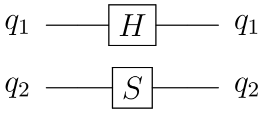
$CNOT\langle Hq_1, Sq_2\rangle.$

Following Proto-Quipper's reduction strategy, the right argument is reduced first, yielding



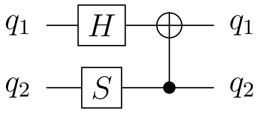
$CNOT\langle Hq_1, q_2\rangle.$

Here we assumed for simplicity that the output wire of the $S$ was not renamed. Since $q_1$ is a value, we now reduce $S q_2$. This yields



$$CNOT\langle q_1, q_2\rangle.$$

Finally, the $CNOT$ gate is applied



$$\langle q_1, q_2\rangle. \tag{8.2}$$

Since $\langle q_1, q_2\rangle$ is a value, the execution is finished. The final circuit is now returned in the form of a term of the language, e.g., as

$$[-, (\langle q_1, q_2\rangle, D, \langle q_1, q_2\rangle)]$$

where $D$ is the constant representing the circuit on the left hand side of (8.2).

# Chapter 9

# Type-safety of Proto-Quipper

In this chapter, we establish that Proto-Quipper is a *type safe* language. As discussed in Section 4.2.4, type safety is established by proving that the language enjoys the subject reduction and progress properties.

## 9.1 Properties of the type system

Before proving the subject reduction and progress, we record some properties of the type system, including the technical but important *Substitution Lemma.* Note that the typing rules enforce a *strict* linearity on variables and quantum variables. In particular, if a quantum variable appears in the quantum context of a valid typing judgement for a term $a$, then it must belong to the free quantum variables of $a$.

**Lemma 9.1.1.**

1. *If* $\Gamma; Q \vdash a : A$ *is valid, then* $Q = FQ(a)$.

2. *If* $\Gamma, x : B; Q \vdash a : A$ *is valid, and* $x \notin FV(a)$*, then* $B = !B'$ *and* $\Gamma; Q \vdash a : A$ *is valid.*

3. *If* $\Gamma; Q \vdash a : A$ *is valid, then* $\Gamma, !\Delta; Q \vdash a : A$ *is valid.*

4. *If* $\Gamma; Q \vdash a : A$ *is valid,* $\Delta <: \Gamma$ *and* $A <: B$*, then* $\Delta; Q \vdash a : B$ *is valid.*

*Proof.* By induction on the corresponding typing derivation. □

**Lemma 9.1.2.** *If* $T$ *is a quantum data type and* $X$ *is a finite subset of* $\mathcal{Q}$*, then* $FQ(\texttt{Spec}_X(T)) \vdash \texttt{Spec}_X(T) : T$ *is valid.*

*Proof.* We prove the Lemma by induction on $T$.

- If $T = 1$, then $\texttt{Spec}_X(T) = *$ and we can use the $(*_i)$ rule.

- If $T = \mathbf{qubit}$, then $\mathtt{Spec}_X(T) = q$ for some quantum variable $q$ and we can use the $(ax_q)$ rule.
- If $T = T_1 \otimes T_2$, then $\mathtt{Spec}_X(T) = \langle t_1, t_2 \rangle$ where $t_1 = \mathtt{Spec}_X(T_1)$ and $u = \mathtt{Spec}_{X \cup \mathrm{FQ}(t_1)}(T_2)$. By the induction hypothesis, both $\mathrm{FQ}(t_1) \vdash t_1 : T_1$ and $\mathrm{FQ}(t_2) \vdash t_2 : T_2$ are valid typing judgements. We can therefore conclude by applying the $(\otimes_i)$ rule. □

**Lemma 9.1.3.** *If $\Gamma; Q \vdash a : A$ is valid and $\mathfrak{b}$ is a binding such that $FQ(a) \subseteq \mathrm{dom}(\mathfrak{b})$ then $\Gamma; \mathfrak{b}(Q) \vdash \mathfrak{b}(a) : A$ is valid.*

*Proof.* By induction on the typing derivation of $\Gamma; Q \vdash a : A$. □

**Lemma 9.1.4.** *If $v \in \mathtt{Val}$ and $\Gamma; Q \vdash v : !A$ is valid, then $Q = \emptyset$ and $\Gamma = !\Delta$ for some $\Delta$.*

*Proof.* By induction on the typing derivation of $\Gamma; Q \vdash v : !A$. In the case of $(ax_c)$, use Lemma 8.2.13.5. □

**Lemma 9.1.5.** *If a term $a$ is not a value then it is of one of the following forms*

- $(t, C, a')$ *with* $a' \notin \mathtt{Val}$,
- $\langle a_1, a_2 \rangle$ *with* $a_1 \notin \mathtt{Val}$ *or* $a_2 \notin \mathtt{Val}$,
- $\mathtt{if}\ a_1\ \mathtt{then}\ a_2\ \mathtt{else}\ a_3$,
- $\mathtt{let}\ * = a_1\ \mathtt{in}\ a_2$,
- $\mathtt{let}\ \langle x, y \rangle = a_1\ \mathtt{in}\ a_2$, *or*
- $a_1 a_2$ *with* $a_1 \neq unbox$ *or* $a_2 \notin \mathtt{Val}$.

*Proof.* By definition of terms and values. □

**Lemma 9.1.6.** *A well-typed value $v$ is either a variable, a quantum variable, a constant or one of the following case occurs*

- *if it is of type* $!^n\ Circ(T, U)$, *it is of the form* $(t, C, u)$ *with* $t$ *and* $u$ *values,*
- *if it is of type* $!^n\mathbf{bool}$ *it is either* $\mathtt{True}$ *or* $\mathtt{False}$,

- *if it is of type* $!^n(A \otimes B)$*, it is of the form* $\langle w, w' \rangle$*, with* $w$ *and* $w'$ *values and* $FQ(w) \cap FQ(w') = \emptyset$*,*
- *if it is of type* $!^n 1$*, it is precisely the term* $*$*, or*
- *if it is of type* $!^n(A \multimap B)$*, it is a lambda abstraction, a constant, or of the form* $unbox(t, C, u)$*.*

*Proof.* By induction on the typing derivation of $v$. □

**Corollary 9.1.7.** *If* $T$ *is a quantum data type and* $v$ *is a well-typed value of type* $T$ *then* $v$ *is a quantum data term.*

**Lemma 9.1.8.** *If* $T$ *is a quantum data type and* $v_1$*,* $v_2$ *are well-typed values of type* $T$*, then* $\mathfrak{b} = bind(v_1, v_2)$ *is a well-defined binding,* $\mathrm{dom}(\mathfrak{b}) = FQ(v_1)$*, and* $\mathrm{cod}(\mathfrak{b}) = FQ(v_2)$*.*

*Proof.* By Corollary 9.1.7, we know that $v_1$ and $v_2$ are quantum data terms so that the statement of the lemma makes sense. The proof then proceeds by induction on $T$, using Lemma 9.1.6. □

**Lemma 9.1.9** (Substitution). *If* $v \in \mathtt{Val}$ *and both* $\Gamma', !\Delta; Q' \vdash v : B$ *and* $\Gamma, !\Delta, x : B; Q \vdash a : A$ *are valid typing judgements, then* $\Gamma, \Gamma', !\Delta; Q, Q' \vdash a[v/x] : A$ *is also valid.*

*Proof.* Let $\pi_1$ and $\pi_2$ be the typing derivations of $\Gamma, !\Delta, x : B; Q \vdash a : A$ and $\Gamma', !\Delta; Q' \vdash v : B$ respectively. We prove the Lemma by induction on $\pi_1$.

- If the last rule of $\pi_1$ is $(ax_c)$ and $a = x$, then $\pi_1$ is

  $$\frac{B <: A}{!\Delta, x : B; \emptyset \vdash x : A}\ (ax_c)$$

  with $\Gamma = Q = \emptyset$. Then $a[v/x] = v$ and can conclude by applying Lemma 9.1.1.4 to $\pi_2$.

- If the last rule of $\pi_1$ is $(ax_c)$ and $a = y \neq x$, then $\pi_1$ is

  $$\frac{A' <: A}{!\Delta, x : !B', y : A'; \emptyset \vdash y : A}\ (ax_c)$$

with $B = !B'$, $Q = \emptyset$ and $\Gamma = \{y : A'\}$ or $\Gamma = \emptyset$ depending on whether or not $A'$ is duplicable. Therefore $v$ is a value of type $!B'$ and by Lemma 9.1.4, we know that $\Gamma' = Q' = \emptyset$. Since $a[v/x] = y$ and $x \notin \mathrm{FV}(y)$ we can conclude by applying Lemma 9.1.1.2 to $\pi_1$.

- If the last rule of $\pi_1$ is one of $(ax_q)$, $(cst)$, $(*_i)$, $(\top)$ and $(\bot)$, and $a$ is the corresponding constant, then $x \notin \mathrm{FV}(a)$ and $x$ must be declared of some type $!B'$. We can therefore reason as in the previous case.

- If the last rule of $\pi_1$ is $(\lambda_1)$ and $a = \lambda y.b$, then $\pi_1$ is

$$\dfrac{\begin{matrix}\vdots \\ \Gamma, !\Delta, x : B, y : A_1; Q \vdash b : A_2\end{matrix}}{\Gamma, !\Delta, x : B; Q \vdash \lambda y.b : A_1 \multimap A_2}\ (\lambda_1)$$

with $A = A_1 \multimap A_2$. By the induction hypothesis, $\Gamma, \Gamma', !\Delta, y : A_1; Q, Q' \vdash b[v/x] : A_2$ is valid and we can conclude by applying $(\lambda_1)$.

- If the last rule of $\pi_1$ is $(\lambda_2)$ and $a = \lambda y.b$, then $\pi_1$ is

$$\dfrac{\begin{matrix}\vdots \\ !\Delta, x : !B', y : A_1; \emptyset \vdash b : A_2\end{matrix}}{!\Delta, x : !B'; \emptyset \vdash \lambda y.b :\ !^{n+1}(A_1 \multimap A_2)}\ (\lambda_2)$$

with $A = !^{n+1}(A_1 \multimap A_2)$ and $B = !B'$. Hence $v$ is a value of type $!B'$ and by Lemma 9.1.4, we know that $\Gamma' = Q' = \emptyset$. The induction hypothesis therefore implies that $!\Delta, y : A_1; \emptyset \vdash b[v/x] : A_2$ is valid and we can conclude by applying $(\lambda_2)$.

- If the last rule of $\pi_1$ is $(app)$, and $a = ca'$, then $\pi_1$ can be of one of three forms depending on $B$. If $B$ is duplicable, then $\pi_1$ is

$$\dfrac{\begin{matrix}\vdots \\ \Gamma_1, x : !B', !\Delta; Q_1 \vdash c : A' \multimap A\end{matrix} \quad \begin{matrix}\vdots \\ \Gamma_2, x : !B', !\Delta; Q_2 \vdash a' : A'\end{matrix}}{\Gamma_1, \Gamma_2, !\Delta, x : !B'; Q_1, Q_2 \vdash ca' : A}\ (app)$$

with $B = !B'$. Using Lemma 9.1.4 again, we know that $\Gamma' = Q' = \emptyset$. The induction hypothesis therefore implies that $\Gamma_1, !\Delta; Q_1 \vdash c[v/x] : A' \multimap A$ and $\Gamma_2, !\Delta; Q_2 \vdash a'[v/x] : A'$ are valid and we can conclude by applying $(app)$. If, instead, $B$ is non-duplicable, then the declaration $x : B$ can only appear in one branch of the derivation. This means that $\pi_1$ is either

$$\dfrac{\overset{\vdots}{\Gamma_1, x : B, !\Delta; Q_1 \vdash c : A' \multimap A} \quad \overset{\vdots}{\Gamma_2, !\Delta; Q_2 \vdash a' : A'}}{\Gamma_1, \Gamma_2, !\Delta, x : B; Q_1, Q_2 \vdash ca' : A}\ (app)$$

or

$$\dfrac{\overset{\vdots}{\Gamma_1, !\Delta; Q_1 \vdash c : A' \multimap A} \quad \overset{\vdots}{\Gamma_2, x : B, !\Delta; Q_2 \vdash a' : A'}}{\Gamma_1, \Gamma_2, !\Delta, x : B; Q_1, Q_2 \vdash ca' : A}\ (app).$$

In the first case, the induction hypothesis implies that $\Gamma_1, \Gamma'!\Delta; Q_1, Q' \vdash c[v/x] : A' \multimap A$ is valid and we can conclude by (*app*). The second case is treated analogously.

- If the last rule of $\pi_1$ is one of $(\otimes_i)$, $(\otimes_e)$, $(*_e)$ and $(if)$, and $a$ is the corresponding term, then we can reason as above by considering in turn the case where $B$ is duplicable and the case where $B$ is non-duplicable.

- If the last rule of $\pi_1$ is (*circ*), and $a = (t, C, a')$, then $\pi_1$ is

$$\dfrac{\overset{\vdots}{Q_1 \vdash t : T} \quad \overset{\vdots}{!\Delta, x : !B'; Q_2 \vdash a' : U} \quad \texttt{In}(C) = Q_1 \quad \texttt{Out}(C) = Q_2}{!\Delta, x : !B'; \emptyset \vdash (t, C, a') : !^n\,\mathrm{Circ}(T, U)}\ (circ)$$

with $A = !^n\,\mathrm{Circ}(T, U)$ and $B = !B'$ for some types $T$, $U$ and $B'$. Using Lemma 9.1.4 again, we know that $\Gamma' = Q' = \emptyset$. The induction hypothesis therefore implies that $!\Gamma; Q_2 \vdash a'[v/x] : U$ is valid and we can conclude by applying (*circ*). □

## 9.2 Subject reduction

We now prove that Proto-Quipper enjoys the subject reduction property. Since the reduction relation is defined on closures but the typing rules apply to terms, we start by extending the notions of typing judgement and validity to closures.

**Definition 9.2.1.** A *typed closure* is an expression of the form:

$$\Gamma; Q \vdash [C, a] : A, (Q'|Q'').$$

It is *valid* if $\texttt{In}(C) = Q'$ and $\texttt{Out}(C) = Q, Q''$, and $\Gamma; Q \vdash a : A$ is a valid typing judgement.

**Lemma 9.2.2.** *If* $[C, a] \to [C', a']$ *then* $\texttt{In}(C) = \texttt{In}(C')$.

*Proof.* By induction on the derivation of $[C, a] \to [C', a']$. In all but the (*unbox*) case, the result follows either from the induction hypothesis or from the fact that $C = C'$. In the (*unbox*) case, use Definition 8.3.1.4a. □

**Theorem 9.2.3** (Subject reduction)**.** *If* $\Gamma; FQ(a) \vdash [C, a] : A, (Q'|Q'')$ *is a valid typed closure and* $[C, a] \to [C', a']$*, then* $\Gamma; FQ(a') \vdash [C', a'] : A, (Q'|Q'')$ *is a valid typed closure.*

*Proof.* We prove the theorem by induction on the derivation of the reduction $[C, a] \to [C', a']$. In each case, we start by reconstructing the unique typing derivation $\pi$ of $\Gamma; \mathrm{FQ}(a) \vdash a : A$ and we use it to prove that $\Gamma; \mathrm{FQ}(a') \vdash [C', a'] : A, (Q'|Q'')$ is valid. By Lemma 9.2.2 we never need to verify that $\texttt{In}(C') = Q'$ so that we only need to show:

- $\texttt{Out}(C') = \mathrm{FQ}(a'), Q''$ and
- $\Gamma; \mathrm{FQ}(a') \vdash a' : A$ is valid.

Throughout the proof, we write $IH(\pi)$ to denote the proof obtained by applying the induction hypothesis to $\pi$.

**Congruence rules:** These rules are treated uniformly. We illustrate the $(fun)$ and $(circ)$ cases.

- $(fun)$: the reduction rule is

$$\frac{[C, c] \to [C', c']}{[C, cb] \to [C', c'b]}$$

with $a = cb$ and $a' = c'b$. The typing derivation $\pi$ is therefore

$$\frac{\overset{\vdots\ \pi_1}{\Gamma_1, !\Delta; \mathrm{FQ}(c) \vdash c : B \multimap A} \quad \overset{\vdots\ \pi_2}{\Gamma_2, !\Delta; \mathrm{FQ}(b) \vdash b : B}}{\Gamma_1, \Gamma_2, !\Delta; \mathrm{FQ}(c), \mathrm{FQ}(b) \vdash cb : A}$$

and $\Gamma_1, \Gamma_2; \mathrm{FQ}(c), \mathrm{FQ}(b) \vdash [C, cb], (Q'|Q'')$ is valid. It follows that

$$\Gamma_1, !\Delta; \mathrm{FQ}(c) \vdash [C, c] : B \multimap A, (Q'|\,\mathrm{FQ}(b), Q'')$$

is valid and, by the induction hypothesis, this implies that $\Gamma_1, !\Delta; \mathrm{FQ}(c') \vdash [C', c'] : B \multimap A, (Q'|\,\mathrm{FQ}(b), Q'')$ is also valid. In particular, it follows that $\mathtt{Out}(C') = \mathrm{FQ}(c'), \mathrm{FQ}(b), Q''$. This, together with the following typing derivation,

$$\dfrac{\begin{array}{c}\vdots\; IH(\pi_1)\\ \Gamma_1, !\Delta; \mathrm{FQ}(c') \vdash c' : B \multimap A\end{array}\quad \begin{array}{c}\vdots\; \pi_2\\ \Gamma_2, !\Delta; \mathrm{FQ}(b) \vdash b : B\end{array}}{\Gamma_1, \Gamma_2, !\Delta; \mathrm{FQ}(c'), \mathrm{FQ}(b) \vdash c'b : A}$$

shows that $\Gamma_1, \Gamma_2, !\Delta; \mathrm{FQ}(c'), \mathrm{FQ}(b) \vdash [C', c'b] : A, (Q', Q'')$ is valid.

- $(circ)$: the reduction rule is

$$\dfrac{[D, b] \to [D', b']}{[C, (t, D, b)] \to [C, (t, D', b')]}\ (circ)$$

with $a = (t, D, b)$ and $a' = (t, D', b')$. The typing derivation $\pi$ is therefore

$$\dfrac{\begin{array}{c}\vdots\; \pi_1\\ \mathrm{FQ}(t) \vdash t : T\end{array}\quad \begin{array}{c}\vdots\; \pi_2\\ !\Delta; \mathrm{FQ}(b) \vdash b : U\end{array}\quad \begin{array}{c}\mathtt{Out}(D) = \mathrm{FQ}(b)\\ \mathtt{In}(D) = \mathrm{FQ}(t)\end{array}}{!\Delta; \emptyset \vdash (t, D, b) : !^n\,\mathrm{Circ}(T, U)}$$

and $!\Delta; \emptyset \vdash [C, (t, D, b)] : !^n\,\mathrm{Circ}(T, U), (Q'|Q'')$ is valid. Disregarding $\pi_1$, it follows from the assumptions in the above rule that $!\Delta; \mathrm{FQ}(b) \vdash [D, b] : U, (\mathrm{FQ}(t)|\emptyset)$ is valid and, by the induction hypothesis, this implies that $!\Delta, \mathrm{FQ}(b') \vdash [D', b'] : U, (\mathrm{FQ}(t)|\emptyset)$ is also valid. This, together with the following typing derivation,

$$\dfrac{\begin{array}{c}\vdots\; \pi_1\\ \mathrm{FQ}(t) \vdash t : T\end{array}\quad \begin{array}{c}\vdots\; IH(\pi_2)\\ !\Delta; \mathrm{FQ}(b') \vdash b' : U\end{array}\quad \begin{array}{c}\mathtt{Out}(D') = \mathrm{FQ}(b')\\ \mathtt{In}(D') = \mathrm{FQ}(t)\end{array}}{!\Delta; \emptyset \vdash (t, D, b') : !^n\,\mathrm{Circ}(T, U)}\ .$$

shows that $!\Delta; \emptyset \vdash [C, (t, D', b')] : !^n\,\mathrm{Circ}(T, U), (Q'|Q'')$ is valid.

**Classical rules:** These rules are also treated uniformly, we illustrate the $(\beta)$ case.

- ($\beta$): the reduction rule is

$$\frac{}{[C, (\lambda x.b)v] \to [C, b[v/x]]}$$

with $a = (\lambda x.b)v$ and $a' = b[v/x]$. The typing derivation $\pi$ is therefore

$$\cfrac{\cfrac{\overset{\vdots\,\pi_1}{\Gamma_1, !\Delta, x : B; \mathrm{FQ}(b) \vdash b : A}}{\Gamma_1, !\Delta; \mathrm{FQ}(b) \vdash \lambda x.b : B \multimap A} \qquad \overset{\vdots\,\pi_2}{\Gamma_2, !\Delta; \mathrm{FQ}(v) \vdash v : B}}{\Gamma_1, \Gamma_2, !\Delta; \mathrm{FQ}(b), \mathrm{FQ}(v) \vdash (\lambda x.b)v : A}$$

and $\Gamma_1, \Gamma_2, !\Delta; \mathrm{FQ}(b), \mathrm{FQ}(v) \vdash [C, (\lambda x.b)v] : A, (Q'|Q'')$ is valid. We then know, by Lemma 9.1.9, that $\Gamma_1, \Gamma_2, !\Delta; \mathrm{FQ}(b), \mathrm{FQ}(v) \vdash b[v/x] : A$ is a valid typing judgement which implies that

$$\Gamma_1, \Gamma_2, !\Delta; \mathrm{FQ}(b), \mathrm{FQ}(v) \vdash [C, b[v/x]] : A, (Q'|Q'')$$

is a valid typed closure.

**Circuit generating rules:** These rules represent the most interesting cases. We treat them individually.

- (*box*): the reduction rule is

$$\frac{\texttt{Spec}(T) = t \quad \texttt{New}(\mathrm{FQ}(t)) = D}{[C, \mathit{box}^T(v)] \to [C, (t, D, vt)]}$$

with $a = \mathit{box}^T(v)$ and $a' = (t, D, vt)$. Since $v$ is a value, we know by Lemma 9.1.4 that the typing derivation $\pi$ is

$$\cfrac{\cfrac{}{!\Delta; \emptyset \vdash \mathit{box}^T : !(T \multimap U) \multimap !^n \mathrm{Circ}(T, U)} \qquad \overset{\vdots\,\pi_1}{!\Delta; \emptyset \vdash v : !(T \multimap U)}}{!\Delta; \emptyset \vdash \mathit{box}^T(v) : !^n \mathrm{Circ}(T, U)}$$

and $!\Delta; \emptyset \vdash [C, \mathit{box}^T(v)] : !^n \mathrm{Circ}(T, U), (Q'|Q'')$ is valid. There exists a typing derivation $\pi_2$ of $\mathrm{FQ}(t) \vdash t : T$, by Lemma 9.1.2. Applying Lemma 9.1.1.4 to $\pi_1$ we get a derivation $\pi_1'$ of $!\Delta; \emptyset \vdash v : T \multimap U$. We can therefore construct the following derivation $\tau$:

$$\cfrac{\overset{\vdots\,\pi_1'}{!\Delta; \emptyset \vdash v : T \multimap U} \quad \overset{\vdots\,\pi_2'}{!\Delta; \mathrm{FQ}(t) \vdash t : T}}{!\Delta; \mathrm{FQ}(t) \vdash vt : U}$$

where $\pi_2'$ is obtained from $\pi_2$ by Lemma 9.1.1.3. Moreover, since $\mathrm{FQ}(vt) = \mathrm{FQ}(t) = \mathtt{Out}(D) = \mathtt{In}(D)$, we have:

$$\dfrac{\dfrac{\vdots\,\pi_2}{\mathrm{FQ}(t) \vdash t : T} \quad \dfrac{\vdots\,\tau}{!\Delta; \mathrm{FQ}(vt) \vdash vt : U} \quad \begin{array}{c}\mathtt{Out}(D) = \mathrm{FQ}(vt)\\ \mathtt{In}(D) = \mathrm{FQ}(t)\end{array}}{!\Delta; \emptyset \vdash (t, D, vt) : !^n \,\mathrm{Circ}(T, U)}\ .$$

Hence $!\Delta; \emptyset \vdash [C, (t, D, vt)] : !^n \,\mathrm{Circ}(T, U)(Q'|Q'')$ is a valid typed closure.

- (*unbox*): the reduction rule is

$$\dfrac{\mathit{bind}(v, u) = \mathfrak{b} \quad \mathtt{Append}(C, D, \mathfrak{b}) = (C', \mathfrak{b}') \quad \mathrm{FQ}(u') \subseteq \mathrm{dom}(\mathfrak{b}')}{[C, (\mathit{unbox}(u, D, u'))v] \to [C', \mathfrak{b}'(u')]}$$

with $a = (\mathit{unbox}(u, D, u'))v$ and $a' = \mathfrak{b}'(u')$. To reconstruct the typing derivation $\pi$, first note that we have the following derivation $\pi_1$ of $!\Delta; \emptyset \vdash \mathit{unbox}(u, D, u') : T \multimap U$

$$\dfrac{\dfrac{}{!\Delta; \emptyset \vdash \mathit{unbox} : \mathrm{Circ}(T, U) \multimap (T \multimap U)} \quad \dfrac{\dfrac{\vdots\,\pi_1^1}{\mathrm{FQ}(u) \vdash u : T} \quad \dfrac{\vdots\,\pi_1^2}{!\Delta; \mathrm{FQ}(u') \vdash u' : U}}{!\Delta; \emptyset \vdash (u, D, u') : \mathrm{Circ}(T, U)}}{!\Delta; \emptyset \vdash \mathit{unbox}(u, D, u') : T \multimap U}$$

with $\mathtt{In}(D) = \mathrm{FQ}(u)$, $\mathtt{Out}(D) = \mathrm{FQ}(u')$. We can then use $\pi_1$ to rebuild $\pi$ as follows:

$$\dfrac{\dfrac{\vdots\,\pi_1}{!\Delta; \emptyset \vdash \mathit{unbox}(u, D, u') : T \multimap U} \quad \dfrac{\vdots\,\pi_2}{!\Delta; \mathrm{FQ}(v) \vdash v : T}}{!\Delta; \mathrm{FQ}(v) \vdash (\mathit{unbox}(u, D, u'))v : U}$$

and the typed closure

$$!\Delta; \mathrm{FQ}(v) \vdash [C, (\mathit{unbox}(u, D, u'))v] : U, (Q'|Q'')$$

is valid. In the conclusion of $\pi_2$, all the term variables are declared of a duplicable type. This follows from Corollary 9.1.7 and Lemma 9.1.1.2. By assumption, we know that $\mathrm{FQ}(u') \subseteq \mathrm{dom}(\mathfrak{b}')$. We can therefore apply Lemma 9.1.3 to $\pi_1^2$ to get a typing derivation $\tau$ of

$$!\Delta; \mathrm{FQ}(\mathfrak{b}'(u')) \vdash \mathfrak{b}(u') : U.$$

Now by Definition 8.3.1.4c we have:

$$\begin{array}{rcl}
\mathtt{Out}(C') & = & \mathfrak{b}'(\mathtt{Out}(D)) \mathbin{\dot\cup} (\mathtt{Out}(C) \setminus \mathfrak{b}^{-1}(\mathtt{In}(D)))\\
& = & \mathfrak{b}'(\mathrm{FQ}(u')) \mathbin{\dot\cup} ((Q'' \mathbin{\dot\cup} \mathrm{FQ}(v)) \setminus \mathfrak{b}^{-1}(\mathrm{FQ}(u)))\\
& = & \mathrm{FQ}(\mathfrak{b}'(u')) \mathbin{\dot\cup} ((Q'' \mathbin{\dot\cup} \mathrm{FQ}(v)) \setminus \mathrm{FQ}(v))\\
& = & \mathrm{FQ}(\mathfrak{b}'(u')) \mathbin{\dot\cup} Q''.
\end{array}$$

Hence $!\Delta; \mathrm{FQ}(\mathfrak{b}(u')) \vdash [C', \mathfrak{b}'(u')] : U, (Q'|Q'')$ is valid.

- $(\mathit{rev})$: the reduction rule is

$$\frac{}{[C, \mathit{rev}\,(t, D, t')] \to [C, (t', D^{-1}, t)]}\ (\mathit{rev})$$

with $a = \mathit{rev}\,(t, D, t')$ and $a' = (t', D^{-1}, t)$. The typing derivation $\pi$ is therefore

$$\dfrac{\dfrac{}{!\Delta; \emptyset \vdash \mathit{rev} : \mathrm{Circ}(T, U) \multimap !^n\, \mathrm{Circ}(U, T)} \qquad \dfrac{\begin{array}{c} \vdots\, \pi_1 \\ \mathrm{FQ}(t) \vdash t : T \end{array} \quad \begin{array}{c} \vdots\, \pi_2 \\ !\Delta; \mathrm{FQ}(t') \vdash t' : U \end{array}}{!\Delta; \emptyset \vdash (t, D, t') : \mathrm{Circ}(T, U)}}{!\Delta; \emptyset \vdash \mathit{rev}\,(t, D, t') : !^n\, \mathrm{Circ}(U, T)}$$

with $\texttt{In}(D) = \mathrm{FQ}(t)$, $\texttt{Out}(D) = \mathrm{FQ}(t')$ and

$$!\Delta; \emptyset \vdash \mathit{rev}\,(t, D, t') : !^n\, \mathrm{Circ}(T, U), (Q'|Q'')$$

is valid. Now note that since $t'$ is a quantum data term, it contains no variables. Applying Lemma 9.1.1.2 to $\pi_2$ repeatedly we therefore get a derivation $\pi_2'$ of $\mathrm{FQ}(t') \vdash t' : U$. Moreover, by applying Lemma 9.1.1.3 to $\pi_1$ we get a typing derivation $\pi_1'$ of $!\Delta, \mathrm{FQ}(t) \vdash t : T$. Since, by Remark 8.3.3, we have $\texttt{Out}(D^{-1}) = \texttt{In}(D) = t$ and $\texttt{In}(D^{-1}) = \texttt{Out}(D) = t'$, we can construct the following typing derivation:

$$\dfrac{\begin{array}{c} \vdots\, \pi_2' \\ \mathrm{FQ}(t') \vdash t' : U \end{array} \quad \begin{array}{c} \vdots\, \pi_1' \\ !\Delta; \mathrm{FQ}(t) \vdash t : T \end{array} \quad \begin{array}{c} \texttt{Out}(D^{-1}) = \mathrm{FQ}(t') \\ \texttt{In}(D^{-1}) = \mathrm{FQ}(t') \end{array}}{!\Delta; \emptyset \vdash (t', D^{-1}, t) : \mathrm{Circ}(U, T)}$$

Hence $!\Delta; \emptyset \vdash (t', D, t) : !^n\, \mathrm{Circ}(T, U), (Q'|Q'')$ is valid. □

**Corollary 9.2.4.** *If $\Gamma; FQ(a) \vdash [C, a] : A, (Q'|Q'')$ is a valid typed closure and $[C, a] \to^* [C', a']$, then $\Gamma; FQ(a') \vdash [C', a'] : A, (Q'|Q'')$ is also a valid typed closure.*

*Proof.* By induction on the length of the reduction sequence. The base case is provided by Theorem 9.2.3. □

The above formulation of Subject Reduction explains why a typed closure contains information about the input and output wires of the circuit state. Indeed, Subject Reduction now guarantees (1) that the input wires of a circuit remain unchanged through reduction and (2) that a term can only affect wires whose identifiers are among its quantum variables.

## 9.3 Progress

We now prove that Proto-Quipper enjoys the progress property.

**Theorem 9.3.1** (Progress)**.** *If $FQ(a) \vdash [C, a] : A, (Q'|Q'')$ is a valid typed closure then either $a \in$ `Val` or there exists a closure $[C', a']$ such that $[C, a] \to [C', a']$.*

First, note that the Progress property is stated for a typed closure whose typing context is empty. This is because the property is not expected to hold if we allow for a non-empty typing context. Indeed, it is easy to see that there are well-typed, non-closed closures such as $[C, xy]$, which are neither values nor reduce. We now prove the theorem.

*Proof.* We prove the theorem by induction on the typing derivation $\pi$ of $\mathrm{FQ}(a) \vdash a : A$. If $a$ is a value then there is nothing to prove. If $a$ is not a value, then by Lemma 9.1.5 there are 6 cases to consider. In each case we show that $[C, a]$ is reducible in the sense that there exists a closure $[C, b]$ such that $[C, a] \to [C, b]$

1. If $a = (t, D, a')$ with $a' \notin \mathtt{Val}$, then the typing derivation $\pi$ is:

$$\dfrac{\begin{array}{c}\vdots\,\pi_1\\ \mathrm{FQ}(t) \vdash t : T\end{array} \quad \begin{array}{c}\vdots\,\pi_2\\ \mathrm{FQ}(a') \vdash a' : U\end{array} \quad \begin{array}{c}\mathtt{Out}(D) = \mathrm{FQ}(a')\\ \mathtt{In}(D) = \mathrm{FQ}(t)\end{array}}{\emptyset \vdash (t, D, a') : \mathrm{Circ}(T, U)}\,.$$

The typed closure

$$\mathrm{FQ}(a') \quad \vdash \quad [D, a'] : U, (\mathrm{FQ}(t)|\emptyset)$$

is therefore valid. Since $a'$ is not a value, the induction hypothesis implies that there exists $a''$ such that $[D, a'] \to [D', a'']$ and $[C, (t, D, a')]$ therefore reduces to $[C, (t, D', a'')]$ by the (*circ*) reduction rule.

2. If $a = \langle a_1, a_2 \rangle$ with $a_1 \notin \mathtt{Val}$ or $a_2 \notin \mathtt{Val}$, then the typing derivation $\pi$ is:

$$\dfrac{\begin{array}{c}\vdots\,\pi_1\\ \mathrm{FQ}(a_1) \vdash a_1 : !^n A_1\end{array} \quad \begin{array}{c}\vdots\,\pi_2\\ \mathrm{FQ}(a_2) \vdash a_2 : !^n A_2\end{array}}{\mathrm{FQ}(a_1), \mathrm{FQ}(a_2) \vdash \langle a_1, a_2 \rangle : !^n (A_1 \otimes A_2)}\,.$$

The typed closures

$$\begin{aligned} \mathrm{FQ}(a_1) &\vdash [C, a_1] : !^n A_1, (Q' |\, \mathrm{FQ}(a_2), Q'') \\ \mathrm{FQ}(a_2) &\vdash [C, a_2] : !^n A_1, (Q' |\, \mathrm{FQ}(a_1), Q'') \end{aligned}$$

are therefore both valid. Now if $a_2 \notin \texttt{Val}$, then by the induction hypothesis $[C, a_2] \to [C', a_2']$. Hence $[C, \langle a_1, a_2 \rangle]$ reduces to $[C', \langle a_1, a_2' \rangle]$ by the (*right*) reduction rule. If on the other hand $a_2 \in \texttt{Val}$, then it must be the case that $a_1 \notin \texttt{Val}$ and we can conclude by reasoning analogously that $[C, \langle a_1, a_2 \rangle]$ reduces to some $[C', \langle a_1', a_2 \rangle]$ by the ($left$) reduction rule.

3. If $a = \texttt{if}\ a_1\ \texttt{then}\ a_2\ \texttt{else}\ a_3$, then the typing derivation $\pi$ is:

$$\frac{\overset{\vdots\, \pi_1}{\mathrm{FQ}(a_1) \vdash a_1 : \mathbf{bool}} \quad \overset{\vdots\, \pi_2}{Q \vdash a_2 : A} \quad \overset{\vdots\, \pi_3}{Q \vdash a_3 : A}}{\mathrm{FQ}(a_1), Q \vdash \texttt{if}\ a_1\ \texttt{then}\ a_2\ \texttt{else}\ a_3 : A}\ .$$

The typed closure

$$\mathrm{FQ}(a_1) \vdash [C, a_1] : \mathbf{bool}, (Q' |\, \mathrm{FQ}(a_2), \mathrm{FQ}(a_3), Q'')$$

is therefore valid. Now if $a_1 \notin \texttt{Val}$, then by the induction hypothesis $[C, a_1] \to [C', a_1']$ and thus $[C, \texttt{if}\ a_1\ \texttt{then}\ a_2\ \texttt{else}\ a_3]$ reduces to $[C', \texttt{if}\ a_1'\ \texttt{then}\ a_2\ \texttt{else}\ a_3]$ by the (*cond*) reduction rule. If on the other hand $a_1 \in \texttt{Val}$, then either $a_1 = \texttt{True}$ or $a_1 = \texttt{False}$, by Lemma 9.1.6. Thus $[C, \texttt{if}\ a_1\ \texttt{then}\ a_2\ \texttt{else}\ a_3]$ reduces either to $[C, a_2]$ by the ($ifT$) reduction rule, or to $[C, a_3]$ by the ($ifF$) reduction rule.

4. If $a = (\texttt{let} * = a_1\ \texttt{in}\ a_2)$, then we can reason as above to show that if $a_1$ is not a value, then the (*let∗*) congruence rule applies, and that if $a_1$ is a value then Lemma 9.1.6 guarantees that the ($*$) rule applies.

5. If $a = (\texttt{let}\ \langle x, y \rangle = a_1\ \texttt{in}\ a_2)$, then we can reason as above to show that if $a_1$ is not a value, then the (*let*) congruence rule applies and that if $a_1$ is a value then Lemma 9.1.6 guarantees that the (*pair*) rule applies.

6. If $a = a_1 a_2$ then the typing derivation $\pi$ is:

$$\frac{\overset{\vdots\, \pi_1}{\mathrm{FQ}(a_1) \vdash a_1 : B \multimap A} \quad \overset{\vdots\, \pi_2}{\mathrm{FQ}(a_2) \vdash a_2 : B}}{\mathrm{FQ}(a_1), \mathrm{FQ}(a_2) \vdash a_1 a_2 : A}\ .$$

The typed closures

$$\mathrm{FQ}(a_1) \;\vdash\; [C, a_1] : B \multimap A, (Q' |\, \mathrm{FQ}(a_2), Q'') \tag{9.1}$$

$$\mathrm{FQ}(a_2) \;\vdash\; [C, a_2] : B, (Q' |\, \mathrm{FQ}(a_1), Q'') \tag{9.2}$$

are therefore valid. There are three cases to treat.

- If $a_1 \notin \mathtt{Val}$, then $[C, a_1 a_2] \to [C', a_1' a_2]$ by the induction hypothesis and the $(fun)$ rule.
- If $a_1 \in \mathtt{Val}$ and $a_2 \notin \mathtt{Val}$, then $[C, a_1 a_2] \to [C', a_1 a_2']$ by the induction hypothesis and the $(arg)$ rule.
- If $a_1, a_2 \in \mathtt{Val}$ then by Lemma 9.1.6, $a_1$ is either an abstraction, a constant, or of the form $\mathit{unbox}(t, C, u)$. If $a_1$ is a lambda abstraction, then $[C, a_1 a_2]$ reduces by the $(\beta)$ rule. If $a_1$ is a constant, then it cannot be *unbox*, since $a_1 a_2$ would then be a value. If $a_1 = \mathit{rev}$, then $a_2$ is a value of type $\mathrm{Circ}(T, U)$. Hence $a_2$ is of the form $(t, C, u)$ by Lemma 9.1.6, so that $[C, a_1 a_2]$ reduces by the (*rev*) rule. If $a_1 = \mathit{box}^T$, then $[C, a_1 a_2]$ reduces by the (*box*) rule.

  Remains to treat the case of $a_1 = \mathit{unbox}(u, D, t)$. For the (*unbox*) rule to apply, we need to show that $\mathit{bind}(a_2, u)$ is well-defined, and that $\mathrm{FQ}(t) \subseteq \mathrm{dom}(\mathfrak{b}')$, where $\mathtt{Append}(C, D, \mathfrak{b}) = (C', \mathfrak{b}')$. The typing derivation $\pi_1$ is the following:

$$\dfrac{\dfrac{}{!\Delta; \emptyset \vdash \mathit{unbox} : \mathrm{Circ}(T, U) \multimap (T \multimap U)} \qquad \dfrac{\overset{\vdots\, \pi_1^1}{\mathrm{FQ}(u) \vdash u : T} \quad \overset{\vdots\, \pi_1^2}{!\Delta; \mathrm{FQ}(t) \vdash t : U}}{!\Delta; \emptyset \vdash (u, D, t) : \mathrm{Circ}(T, U)}}{!\Delta; \emptyset \vdash \mathit{unbox}(u, D, t) : T \multimap U}$$

  with $\mathtt{In}(D) = \mathrm{FQ}(u)$, $\mathtt{Out}(D) = \mathrm{FQ}(t)$, $A = U$ and $B = T$ for some quantum data types $T, U$. It follows that $a_2$ and $u$ are two values of type $T$, so that, by Lemma 9.1.8, $\mathfrak{b} = \mathit{bind}(a_2, u)$ is defined with $\mathrm{dom}(\mathfrak{b}) = \mathrm{FQ}(a_2)$ and $\mathrm{cod}(\mathfrak{b}) = \mathrm{FQ}(u)$. Moreover, $\mathrm{FQ}(a_2) \subseteq \mathtt{Out}(C)$ by the validity of the typed closure (9.2), and $\mathrm{FQ}(u) = \mathtt{In}(D)$ as noted above. Therefore, $\mathrm{dom}(\mathfrak{b}) \subseteq \mathtt{Out}(C)$ and $\mathrm{cod}(\mathfrak{b}) = \mathtt{In}(D)$ hold, as required by Definition 8.3.1.4b. By Definition 8.3.1.4b, we conclude that $\mathrm{dom}(\mathfrak{b}') = \mathtt{Out}(D) = \mathrm{FQ}(t)$, so that the (*unbox*) rule in fact applies. □

# Chapter 10

# Conclusion

In this thesis, we applied tools from algebraic number theory and mathematical logic to problems in the theory of quantum computation. We described algorithms to solve the problem of approximate synthesis of special unitaries over the Clifford+$V$ and Clifford+$T$ gate sets. We also defined a typed lambda calculus for quantum computation called Proto-Quipper which serves as a mathematical foundation for the Quipper quantum programming language. In conclusion, we briefly describe some avenues for future research.

## 10.1 Approximate synthesis

The synthesis methods described in chapters 6 and 7 belong to a very recent family of number-theoretic algorithms. Many generalization of these methods can be considered.

- The algorithms described in chapters 6 and 7 are only optimal for $z$-rotations. While Euler angle decompositions can be used to extend these methods to arbitrary special unitaries, optimality is lost in the process. A first potential generalization of the methods of chapters 6 and 7 is to define optimal number-theoretic synthesis algorithms for arbitrary special unitaries.

- A second restriction of the decomposition methods presented in chapters 6 and 7 is that they are only defined for specific gate sets, namely the Clifford+$V$ and Clifford+$T$ gate sets. Another future generalization of this work is to extend the number-theoretic synthesis methods to different gate sets. This line of enquiry has already been pursued in the recent literature with encouraging results. In particular, it is known that asymptotically optimal number-theoretic decomposition methods can be defined for certain gate sets based on anyonic braidings (see [39] and [6]). Further, exact synthesis methods have recently been devised for a relatively general family of gate sets ([19] and [43]). While these exact synthesis methods have not yet been extended to approximate synthesis algorithms, we expect that, at least in some cases, this extension should carry through with relative ease.

- A further possible generalization is to consider unitary groups in higher dimensions. Higher-dimensional versions of the Clifford gates have previously been studied in the literature [25]. Moreover, higher-dimensional analogues of the $T$ gate were recently introduced [33]. Together, these define a higher-dimensional Clifford+$T$ gate set which stands as a natural candidate for an adaptation of the methods of chapters 6 and 7.

We note that the decomposition algorithm of Fowler [20] as well as the Solovay-Kitaev algorithm [14] are both very general. Indeed, both algorithms allow the synthesis of arbitrary special unitaries over any gate set and in any dimension. Since the algorithms presented in chapters 6 and 7 rely on specific properties of the rings of algebraic integers associated with the Clifford+$V$ and Clifford+$T$ gate set, there is no reason for these methods to generalize to arbitrary gate sets. However, it might be possible to identify a general class of gate sets to which these methods apply.

Another interesting avenue of future research lies in a modification of the statement of the synthesis problem itself, by allowing a broader notion of circuit. As a first such generalization, one can introduce *ancillary* qubits in the approximating circuit. Suppose a special unitary $U$ and a precision $\varepsilon > 0$ are given. Instead of searching for $W \in \mathrm{U}(2)$ such that $\|U - W\| < \varepsilon$ one can look for $W \in \mathrm{U}(2^{n+1})$ such that for any state $|\phi\rangle$ we have

$$\|U(|\phi\rangle)|0\ldots0\rangle - W(|\phi\rangle|0\ldots0\rangle)\| < \varepsilon.$$

In other words, unitaries acting on more than one qubit can be considered, provided that they return the additional qubits nearly unchanged. The advantage of such a generalized notion of circuit is that it opens the door to a certain form of parallelization. In particular, even if the number of non-Clifford gates in the approximating circuit remains unchanged, applying them in parallel, rather than sequentially, may represent a gain.

Another generalization of the synthesis problem is to allow the approximating circuits to use measurements or other adaptive methods. It is known that using such methods can decrease the gate count below the information-theoretic lower bound. These methods are relatively new, but very promising results have already been achieved ([66], [51], [8], and [67]).

## 10.2 Proto-Quipper

As already mentioned in Chapter 1, the rationale behind the design of Proto-Quipper was to start with the simplest language possible, establish type-safety, and then extend the language in small steps with the goal of eventually adding most of Quipper's features to Proto-Quipper in a type-safe way. This defines a natural set of problems for future work. Many such extensions are conceivable, but we only describe a few here.

- In the current version of Proto-Quipper, all circuits are reversible. This follows from the definition of the (*rev*) and (*circ*) typing rules and will have to be modified to accommodate non-reversible gates such as measurements. In such a setting, the type system should ensure that circuits are reversed only if it is meaningful to do so. In particular, if a circuit contains a measurement, then it should not be possible to reverse it.

- A circuit generating function that inputs a list of qubits does not define just one circuit, but rather a family of circuits parameterized by the length $n$ of the list. To box such a function, a particular value of $n$ has to be given. In Quipper, we refer to $n$ as the "shape" of the argument of the function. Operations such as boxing and reversing often require shape information. An alternative solution would be to equip Proto-Quipper with a *dependent type system*. This would allow shape information to be stored at the type level.

- In contrast to the quantum lambda calculus, the reduction in Proto-Quipper is non-probabilistic. Of course, the hypothetical quantum device running the circuit produced by Proto-Quipper would have to perform probabilistic operations, but the circuit generation itself does not have to. Even if the language is extended with measurement gates, it is still possible to generate the circuits deterministically. This is justified by the "principle of deferred measurement" which states that any quantum circuit is equivalent to one where all measurements are performed as the very last operations (see, e.g., [50] p.186). We therefore do not need to rely on the result of a measurement to construct circuits and, in theory, no computational power is lost by making this assumption. In practice, however, this delaying of measurement may significantly increase the size of the circuit. Thus in terms of computational resources it is sometimes advantageous to permit circuit generating functions that access previous measurement results. Several existing quantum algorithms rely on such interactive circuit building. In Quipper, this capability is captured by the notion of *dynamic lifting*. Adding such a feature to Proto-Quipper would make the reduction relation probabilistic. It is an interesting research problem how such an extension can be carried out in a type-safe way.

# Bibliography

[1] D.S. Alexander, Neil J. Ross, P. Selinger, J.M. Smith, and B. Valiron. Programming the quantum future. *Communications of the ACM*, 2015. To appear.

[2] A. Ambainis, A. M. Childs, B.W. Reichardt, R. Špalek, and S. Zhang. Any AND-OR formula of size $n$ can be evaluated in time $n^{\frac{1}{2}+o(1)}$ on a quantum computer. *SIAM J. Comput.*, 39:2513–2530, 2010.

[3] Pablo Arrighi and Gilles Dowek. Linear-algebraic lambda-calculus: higher-order, encodings, and confluence. In *Proceedings of the 19th international conference on Rewriting Techniques and Applications*, volume 5117 of *Lecture Notes in Computer Science*, page 1731, 2008.

[4] H.P. Barendregt. *The Lambda Calculus : Its Syntax and Semantics.*, volume 103 of *Studies in Logic and the Foundations of Mathematics.* MIT Press, 1981.

[5] Andreas Blass, Alex Bocharov, and Yuri Gurevich. Optimal ancilla-free Pauli+$V$ circuits for axial rotations. Available from `arXiv:1412.1033`, December 2014.

[6] Alex Bocharov, Xingshan Cui, Vadym Kliuchnikov, and Zhenghan Wang. Efficient topological compilation for weakly-integral anyon model. Available from `arXiv:1504.03383`, April 2015.

[7] Alex Bocharov, Yuri Gurevich, and Krysta M. Svore. Efficient decomposition of single-qubit gates into $V$ basis circuits. *Phys. Rev. A*, 88:012313 (13 pages), 2013. Also available from `arXiv:1303.1411`.

[8] Alex Bocharov, Martin Roetteler, and Krysta M. Svore. Efficient synthesis of universal repeat-until-success circuits. Available from `arXiv:1404.5320`, April 2014.

[9] H. Chataing, N. J. Ross, and P. Selinger. Report on Proto-Quipper 0.2. Unpublished report delivered to IARPA in the context of the QCS project, 2013.

[10] A. M. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann, and D. A. Spielman. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, pages 59–68, 2003.

[11] A. Church and J. B. Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39:472–482, 1936.

[12] Koen Claessen. *Embedded Languages for Describing and Verifying Hardware.* PhD thesis, Chalmers University of Technology and Göteborg University, 2001.

[13] Henri Cohen. *Advanced topics in computational number theory.* Graduate texts in mathematics. Springer, New York, N.Y., Berlin, Heidelberg,, 2000.

[14] Christopher M. Dawson and Michael A. Nielsen. The Solovay-Kitaev algorithm. *Quantum Information and Computation*, 6(1):81–95, January 2006. Also available from `arXiv:quant-ph/0505030`.

[15] David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London, Series A, Mathematical and Physical Sciences*, 400:97–117, 1985.

[16] David Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London, Series A, Mathematical and Physical Sciences*, 425(1868):73–90, 1989.

[17] M. Felleisen and A. K. Wright. A syntactic approach to type soundness. *Information and Computation*, pages 38–94, November 1994.

[18] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6-7):467–488, June 1982.

[19] Simon Forest, David Gosset, Vadym Kliuchnikov, and David McKinnon. Exact synthesis of single-qubit unitaries over Clifford-cyclotomic gate sets. Available from `arXiv:1501.4150`, January 2015.

[20] Austin G. Fowler. Constructing arbitrary Steane code single logical qubit fault-tolerant gates. *Quantum Information and Computation*, 11(9–10):867–873, 2011. Also available from `arXiv:quant-ph/0411206`.

[21] Brett Giles and Peter Selinger. Exact synthesis of multiqubit Clifford+$T$ circuits. *Physical Review A*, 87:032332, 2013. Also available from `arXiv:1212.0506`.

[22] Brett Giles and Peter Selinger. Remarks on Matsumoto and Amano's normal form for single-qubit Clifford+$T$ operators. Available from `arXiv:1312.6584`, December 2013.

[23] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[24] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types.* Number 7 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1989.

[25] Daniel Gottesman. Fault-tolerant quantum computation with higher-dimensional systems. In *Quantum Computing and Quantum Communications, Proceedings of the 1st NASA International Conference on Quantum Computing and Quantum Communications (QCQC)*, pages 302–313, New York, NY, USA, 1998. Springer-Verlag.

[26] Daniel Gottesman. The Heisenberg representation of quantum computers. In *International Conference on Group Theoretic Methods in Physics*, page 9807006, 1998.

[27] Alexander Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. An introduction to quantum programming in Quipper. In *Proceedings of the 5th International Conference on Reversible Computation*, volume 7948 of *Lecture Notes in Computer Science*, pages 110–124, 2013. Preprint available from `arXiv:1304.5485`.

[28] Alexander Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: a scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 333–342, 2013. Preprint available from `arXiv:1304.3390`.

[29] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. ACM.

[30] Sean Hallgren. Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. *J. ACM*, 54(1):4:1–4:19, March 2007.

[31] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, 103(15):150502, 2009.

[32] A.W. Harrow, B. Recht, and I.L. Chuang. Efficient discrete approximations of quantum gates. *Journal of Mathematical Physics*, 43, 2002. Also available from `arXiv:quant-ph/0111031`.

[33] Mark Howard and Jiri Vala. Qudit versions of the qubit $\pi/8$ gate. *Phys. Rev. A*, 86:022316, Aug 2012.

[34] Graham Hutton. *Programming in Haskell*. Cambridge University Press, January 2007.

[35] IARPA Quantum Computer Science Program. Broad Agency Announcement IARPA-BAA-10-02. Available from `https://www.fbo.gov/notices/637e87ac1274d030ce2ab69339ccf93c`, April 2010.

[36] Phillip Kaye, Raymond Laflamme, and Michele Mosca. *An Introduction to Quantum Computing*. Oxford University Press, Inc., New York, NY, USA, 2007.

[37] A. Yu. Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):11911249, 1997.

[38] A. Yu. Kitaev, A. H. Shen, and M. N. Vyalyi. *Classical and Quantum Computation*. Graduate Studies in Mathematics 47. American Mathematical Society, 2002.

[39] Vadym Kliuchnikov, Alex Bocharov, and Krysta M. Svore. Asymptotically optimal topological quantum compiling. Available from arXiv:1310.4150, October 2013.

[40] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Practical approximation of single-qubit unitaries by single-qubit quantum Clifford and $T$ circuits. Also available from arXiv:1212.6964, December 2012.

[41] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Asymptotically optimal approximation of single qubit unitaries by Clifford and $T$ circuits using a constant number of ancillary qubits. *Phys. Rev. Lett.*, 110:190502 (5 pages), 2013. Also available from arXiv:1212.0822v2.

[42] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Fast and efficient exact synthesis of single qubit unitaries generated by Clifford and $T$ gates. *Quantum Information and Computation*, 13(7–8):607–630, 2013. Also available from arXiv:1206.5236v4.

[43] Vadym Kliuchnikov and Jon Yard. A framework for exact synthesis. Available from arXiv:1504.04350, April 2015.

[44] E. Knill. Conventions for quantum pseudocode, 1996.

[45] H.W. Lenstra, Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8:538 – 548, 1983.

[46] A. Lubotzky, R. Phillips, and P. Sarnak. Hecke operators and distributing points on the sphere I. *Communications on Pure and Applied Mathematics*, 39:S149–S186, 1986.

[47] A. Lubotzky, R. Phillips, and P. Sarnak. Hecke operators and distributing points on $S^2$ II. *Communications on Pure and Applied Mathematics*, 40:401–420, 1987.

[48] Frédéric Magniez, Miklos Santha, and Mario Szegedy. Quantum algorithms for the triangle problem. quant-ph/0310134, 2003.

[49] Wladyslaw Narkiewicz. *Elementary and analytic theory of algebraic numbers.* Springer-Verlag Warszawa, Berlin, 1990.

[50] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2002.

[51] Adam Paetznick and Krysta M. Svore. Repeat-until-success: Non-deterministic decomposition of single-qubit unitaries. Available from arXiv:1311.1074, November 2013.

[52] Robert Raussendorf and Hans J. Briegel. Computational model underlying the one-way quantum computer. *Quantum Info. Comput.*, 2(6):443–486, October 2002.

[53] Oded Regev. Quantum computation and lattice problems. *SIAM J. Comput.*, 33(3):738–760, 2004.

[54] Neil J. Ross. Optimal ancilla-free Clifford+$V$ approximation of $z$-rotations. *Quantum Information and Computation*, 15(11–12):932–950, 2015. Preprint available from `arXiv:1409.4355`.

[55] Neil J. Ross, P. Selinger, J.M. Smith, and B. Valiron. Quipper: Concrete resource estimation in quantum algorithms. Extended abstract for QAPL 2014. Available from `arXiv:1412.0625`, 2014.

[56] Neil J. Ross and Peter Selinger. Optimal ancilla-free Clifford+$T$ approximation of $z$-rotations. Available from `arXiv:1403.2975`, March 2014.

[57] Neil J. Ross and Peter Selinger. Exact and approximate synthesis of quantum circuits, version 0.3.0.1. Software implementation available from `http://www.mathstat.dal.ca/~selinger/newsynth/`, 2015.

[58] P. Selinger and B. Valiron. Quantum lambda calculus. In S. Gay and I. Mackie, editors, *Semantic Techniques in Quantum Computation*, pages 135–172. Cambridge University Press, 2009.

[59] Peter Selinger. Towards a quantum programming language. *Mathematical. Structures in Comp. Sci.*, 14(4):527–586, August 2004.

[60] Peter Selinger. Efficient Clifford+$T$ approximation of single-qubit operators. *Quantum Information and Computation*, 15(1–2):159–180, 2015. Preprint available from `arXiv:1212.6253`.

[61] Peter Selinger and Benoît Valiron. A lambda calculus for quantum computation with classical control. *Mathematical Structures in Computer Science*, 16(3):527–552, 2006.

[62] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. Also available from `arXiv:quant-ph/9508027`.

[63] André van Tonder. A lambda calculus for quantum computation. *SIAM J. Comput.*, 33(5):1109–1135, May 2004.

[64] Benoît Valiron. A functional programming language for quantum computation with classical control. Master's thesis, University of Ottawa, September 2004.

[65] James D. Whitfield, Jacob Biamonte, and Alán Aspuru-Guzik. Simulation of electronic structure Hamiltonians using quantum computers. *Molecular Physics*, 109(5):735–750, 2011.

[66] Nathan Wiebe and Vadym Kliuchnikov. Floating point representations in quantum circuit synthesis. Available from arXiv:1305.5528, May 2013.

[67] Nathan Wiebe and Martin Roetteler. Quantum arithmetic and numerical analysis using repeat-until-success circuits. Available from arXiv:1406.2040, June 2014.