TU WIEN Informatics

Eine Referenzimplementierung für den erweiterten Algorithmus von Simon

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Oskar Mayer, Bsc
Matrikelnummer 01426798

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Ao. Univ-Prof. Dr. Uwe Egly

Wien, 2. Mai 2025

Oskar Mayer	Uwe Egly

TU WIEN Informatics

A Reference Implementation for the Extended Version of Simon's Algorithm

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Oskar Mayer, Bsc
Registration Number 01426798

to the Faculty of Informatics

at the TU Wien

Advisor: Ao. Univ-Prof. Dr. Uwe Egly

Vienna, May 2, 2025

Oskar Mayer	Uwe Egly

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Oskar Mayer, Bsc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, habe ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT-Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 2. Mai 2025

Oskar Mayer

Acknowledgements

I am most grateful to Uwe Egly for guiding the entire process of writing this thesis. In particular, I am thankful for the diligent proofreading, helpful discussions, and all the valuable feedback which made working on this thesis a pleasant experience from start to finish.

Writing this thesis was also supported by the colleagues and friends I made at TU Wien. Thank you to Felix and Paul for cheering up countless study sessions at the library. Greetings to Tabea, who talked me into signing up for my first course on quantum computing. Thanks also to Alice for patiently correcting my English writing.

Lijep pozdrav Ani. Volim te najviše!

Kurzfassung

Simons Problem und *Simons Algorithmus* sind fundamentale Ergebnisse aus dem Bereich Quantum Computing. Im Jahr 1997 publizierten Brassard und Høyer eine Erweiterung der Arbeit von Simon, welche als die *erweiterte Version von Simons Problem und Algorithmus* bekannt ist, und welche schließlich zur Entwicklung zusätzlicher wichtiger Konzepte wie dem *Problem der verborgenen Untergruppe* und dem *Amplitudenverstärkungsverfahren* führte. Trotz ihres Einflusses existierte die Arbeit von Brassard und Høyer bis jetzt nur auf dem Papier und es gab keine Referenzimplementierung für den erweiterten Algorithmus von Simon.

Im Zuge dieser Diplomarbeit führen wir zunächst eine detaillierte mathematische Analyse der Arbeit Brassard- und Høyers durch. Weiters stellen wir eine neuartige Prozedur vor, mit welcher automatisch Testinstanzen für die erweiterte Version von Simons Problem generiert werden können. Bei diesen Testinstanzen handelt es sich um Quantenprogramme, welche gemeinhin *Orakel* genannt werden. Unsere Orakel sind auf real existierenden Quantencomputern lauffähig. Darüber hinaus präsentieren wir die erste Referenzimplementierung der erweiterten Version von Simons Algorithmus, welche ebenfalls auf echten Quantencomputern lauffähig ist.

Abschließend evaluieren wir unsere Implementierung auf einem störungsfreien Quantencomputersimulator, auf mehreren störungsbehafteten Simulatoren echter Geräte und auf einem real existierenden Quantencomputer selbst. Unsere Erkenntnisse sind, dass sich unsere Implementierung sowohl auf dem störungsfreien- als auch auf den störungsbehafteten Simulatoren verhält wie erwartet. Die Ergebnisse auf dem echten Quantencomputer legen jedoch nahe, dass die momentan verfügbare Quantenhardware nicht ausgereift genug ist, um selbst kleinste Instanzen der erweiterten Version von Simons Problem zu lösen.

Abstract

Simon's problem and *Simon's algorithm* are seminal results in quantum computing. In 1997, Brassard and Høyer published an extension to the work of Simon known as the *extended version of Simon's problem and algorithm*, which itself laid the foundations for further important concepts like the *hidden subgroup problem* framework and the *amplitude amplification* quantum algorithm paradigm. Despite this, the work of Brassard and Høyer so far existed on paper only and lacked a reference implementation.

We first provide a detailed mathematical analysis of the work by Brassard and Høyer. Second, we present a novel procedure that automatically generates test instances for the extended version of Simon's problem. Those test instances are quantum programs commonly called *oracles*, and our oracles are runnable on actual quantum hardware. Third, we provide the first reference implementation for the extended version of Simon's algorithm itself, which is runnable on actual quantum hardware as well.

Last, we tested our implementation on a noise-free quantum computer simulator, on noisy simulators for actual quantum processors and on a real quantum device. Our findings are that the extended version of Simon's algorithm works as expected both on the noise-free and the noisy simulators. The results from the real quantum computer however hint that current-generation devices are still not mature enough to solve even small toy instances of the extended version of Simon's problem.

Contents

CHAPTER 1

Introduction

Imagine we are given some kind of mysterious machine. The machine is sealed shut, so we cannot take it apart and inspect how it works on the inside. However, we can turn the machine on and interact with it. That is, we can feed the machine inputs and it will give us outputs in return. We are also given a user manual claiming that the machine fulfills exactly one of the following conditions: Either each possible input gets mapped to a unique output, or there are always two distinct inputs that get mapped to the same output. Our job is to find out whether the machine fulfills the first or the second condition and, in the latter case, what are the pairs of inputs that get mapped to the same output.

The setting from above is better known as *Simon's problem* [32]. In its formal description, instead of an actual physical machine, we are given a black box implementation of some function which we can run on a computer. We immediately note that a trivial way to solve Simon's problem is to simply run the black box function once for each possible input and to log the results. However, in the case of Simon's problem, the input space of the function are bitstrings of length n, so we would be required to make $O(2^n)$ queries to the function, too much to be feasible for large n. More sophisticated solutions exist, but even state-of-the-art algorithms (deterministic as well as randomized) require an amount of queries exponential in n [32].

Although Simon's problem is rather abstract and its usefulness is not immediately obvious, its proposal in 1994 caused excitement in the field of quantum computing. Simon's problem is the first problem ever discovered where, if we use a quantum computer, we can find a solution substantially faster than we could using classical hardware only. The corresponding hybrid quantum-classical algorithm is now known as *Simon's algorithm* [32], and it is considered a seminal result, covered in most quantum computing textbooks and courses. Importantly, Simon's algorithm is probabilistic, that is, we do not have a guaranteed upper bound for its runtime, only an expected one.

Over time, both Simon's problem and Simon's algorithm have been generalized and adapted. In 1997, Brassard and Høyer presented what is now called the *extended version of Simon's problem* [13]. In that extended version, we are given as problem instance a black box function implementation which does not fulfill one of two, but one of multiple (i.e. possibly more than two) conditions. The task is again to find out, which condition the black box implementation fulfills. Additionally, Brassard and Høyer presented what is now called the *extended version of Simon's algorithm* [13]. It not only solves the more general extended version of Simon's problem, it further does so deterministically (with a fixed upper runtime bound) and using the same amount of queries against the black box.

The extended version of Simon's problem was eventually generalized into what is now commonly called the hidden subgroup problem (HSP) [27]. In this most general definition we are still given a black box implementation of some function. The input space of that function is not comprised of bitstrings of length n anymore, but of elements of some algebraic group G, and the behavior of the function (i.e. which input values are mapped to which output values) is specified via the structure of a 'hidden' subgroup of G. While the previous versions of Simon's problem serve mainly theoretical purposes, important real-world applications have reductions to the HSP. For example, the celebrated algorithm of Shor [31], for computing the prime factor decomposition of a number, can be seen as an instance of the HSP where G is the algebraic group over the set of integers where the group operation is integer addition [27].

The extended version of Simon's algorithm has received continuous attention since its original proposal as well. The technique at the heart of the algorithm is now called *amplitude amplification* [14, 18], and it is often used to speed up the search in an unstructured search space. An example use case is attacking symmetric cryptographic primitives [15], like inverting a hash function.

Although the work by Brassard and Høyer [13] proved to be so influential, no runnable implementation of the extended version of Simon's algorithm was ever published. A reason for this is that back in 1997, when Brassard and Høyer proposed their work, actual quantum computers were, aside from primitive experimental prototypes, merely science-fiction. Hence, developing a quantum algorithm meant giving a written description of abstract mathematical operations being executed on some hypothetical device, and then proving the correctness of the entire procedure on paper. The absence of actual quantum computers meant there was no use developing concrete runnable implementations of quantum algorithms. This situation has changed. At the time of writing this thesis, commercial quantum computers are available, for example IBM Eagle [4] or IonQ Aria [5]. Since quantum hardware is now a reality, we would also like to experiment with the algorithms that so far exist on paper only, and this is the research gap we are trying to fill. Our contributions are:

- A rigorous mathematical analysis of the work of Brassard and Høyer [13]. We formally relate the original version of Simon's problem to the extended version and we give detailed proofs for the correctness and runtime bounds of the extended

version of Simon's algorithm. Analyses like this already exist, but they generally stay on a high level and tend to skip important details and edge cases.

- A novel procedure for automatically generating test instances for the extended version of Simon's problem, which we implemented using the IBM Qiskit [21] quantum SDK. Such a procedure already exists for the original version of Simon's problem, but not for the extended version.
- A runnable reference implementation [24] for the extended version of Simon's algorithm as presented by Brassard and Høyer [13], also using the IBM Qiskit SDK. To the best of our knowledge, our reference implementation is the first one ever published.

The rest of this thesis is structured as follows. We first give an overview over formalisms in quantum computing. Chapter 2 contains core mathematical concepts and notation used throughout the thesis. In Chapter 3 we give a short introduction to quantum complexity theory. We need this in order to formalize what it means for an algorithm to be efficient in the context of the extended version of Simon's problem.

Next, we give a detailed description of the work of Simon [32]. Chapter 4 contains a formal definition of the original version of Simon's problem, together with a detailed analysis of the original version of Simon's algorithm.

We follow this up with a detailed description of the work by Brassard and Høyer [13]. In Chapter 5 we first present a formal definition of the extended version of Simon's problem. We then repeat the analysis of the original version of Simon's algorithm, but now in the setting of the extended version of Simon's problem. In Chapters 6 and 7 we present the techniques with which we can make the original version of Simon's algorithm deterministic and fit for the extended version of Simon's problem. In Chapter 8, we tie everything together and give a formal description of the extended version of Simon's algorithm.

Finally, we present and discuss our implementation in Chapter 9. We first show the procedure to generate test instances for the extended version of Simon's problem. Next, we present examples for the circuits generated by our overall implementation of the extended version of Simon's algorithm, where we show which parts of the circuits implement which parts of the quantum algorithm. We then present experimental results from running those circuits both on simulators and on an actual IBM Eagle [4] processor. Chapter 10 contains concluding remarks.

CHAPTER 2

Notation and Mathematical Background

In this section we first present core concepts from group theory needed for the analysis of the extended version of Simon's problem. Next, we present general results from linear algebra. Those are needed first for a better understanding of how quantum computing works. Second, throughout this thesis, we will work with bitstrings a lot, and it will often be helpful to view those as a vector space over $\mathbb{Z}_2$ to derive more advanced results. Next we present linear algebra results special to quantum computing and last we give an overview on the concrete quantum operators used in the thesis.

2.1 Group Theory

Following Appendix 2 in [27] by Nielsen and Chuang, a *group* is a tuple $\langle X, \circ \rangle$ where X is some non-empty set and $\circ$ denotes a binary operation with the following properties.

- $\forall x, y \in X : (x \circ y) \in X$ (closure)
- $\forall x, y, z \in X : (x \circ y) \circ z = x \circ (y \circ z)$ (associativity)
- $\exists e \in X : \forall x \in X : x \circ e = e \circ x = x$ (neutral element)
- $\forall x \in X : \exists x^{-1} \in X : x \circ x^{-1} = x^{-1} \circ x = e$ (inverses)

As a shorthand, we often refer to a group $\langle X, \circ \rangle$ by just X when the corresponding binary operation is clear from the context.

We now follow Brassard and Høyer [13] as we present the particular group G defined as $\langle \{0,1\}^n, \oplus \rangle$, where

- $\{0,1\}^n$ denotes the set of all possible bitstrings of length n, $n > 0$ and
- $\oplus$ denotes the bitwise XOR operation.

The neutral element of G is the zero bitstring, which we denote as $\mathbf{0}$. Furthermore, in G, each element is its own inverse. For any $H \subseteq \{0,1\}^n$, if $\langle H, \oplus \rangle$ is a group, we call H a *subgroup* of G. If H additionally contains an element that is not the zero bitstring, we call H a *non-trivial* subgroup of G.

For a subgroup H of G and any arbitrary $y \in G$, the *coset* of H induced by y is defined as $\{y \oplus h \mid h \in H\}$, and we call y the *representative* of that coset. Furthermore, we write $y \oplus H$ as a shorthand for the set $\{y \oplus h \mid h \in H\}$.

If Y is a subset of G, then $\langle Y \rangle$ denotes the set *generated* by Y. More formally, we have

$$\langle Y \rangle = \begin{cases} \{\mathbf{0}\} & Y = \emptyset \\ \{x \mid \exists y^{(1)}, \dots y^{(j)} \in Y : x = y^{(1)} \oplus \dots \oplus y^{(j)}\} & \text{otherwise} \end{cases}.$$

For any $x \in G$ and for any i with $0 \leq i < n$, we write x_i to refer to the bit at index i in x. We now deviate from the notation given by Brassard and Høyer [13] by introducing a separate symbol for the bitwise inner product modulo 2 between two bitstrings

The $\odot$ symbol denotes the *bitwise inner product modulo 2* of two bitstrings $x, y \in G$. Formally it is defined as

$$x \odot y = (x_{n-1} \cdot y_{n-1}) \oplus (x_{n-2} \cdot y_{n-2}) \oplus \dots \oplus (x_0 \cdot y_0).$$

Note that Brassard and Høyer do not use the $\odot$ symbol in their paper [13]. They instead overload the $\cdot$ symbol and use it for both the bitwise inner product and for regular integer multiplication.

For any set $Y \subseteq G$, we define

$$Y^\perp = \{g \mid g \in G, \forall y \in Y : g \odot y = 0\},$$

and we call $Y^\perp$ the *orthogonal* set of Y.

2.2 Linear Algebra Basics

It is outside the scope of this work to provide either a comprehensive introduction to the field of linear algebra or to the field of quantum computing. We will however formally introduce some concepts particularly useful for the analysis of the algorithms that we present here. We loosely follow Section 2.1 in [27] by Nielsen and Chuang.

For any complex number a, let $\overline{a}$ denote its complex conjugate. Let V be a vector space of dimension n. As is common in quantum computing, when we talk about the basis of

V, we refer to the *computational standard basis*

$$n\left\{\begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix}, \begin{pmatrix}0\\1\\\vdots\\0\end{pmatrix}, \ \cdots\ , \begin{pmatrix}0\\0\\\vdots\\1\end{pmatrix}\right. .$$

In general, all vector spaces referenced in this work are considered to be finite-dimensional. Consider a function $\langle\cdot|\cdot\rangle$ with $\langle\cdot|\cdot\rangle : (V \times V) \to \mathbb{C}$. We call $\langle\cdot|\cdot\rangle$ an *inner product* (or *inner product function*) if $\forall x, y, z \in V$ and $\forall \lambda, \mu \in \mathbb{C}$ we have

- $\langle x|\lambda y + \mu z\rangle = \lambda \langle x|y\rangle + \mu \langle x|z\rangle$,
- $\langle x|y\rangle = \overline{\langle y|x\rangle}$,
- $\langle x|x\rangle \geq 0$ and $\langle x|x\rangle = 0$ if and only if x is the zero vector.

The pair $(V, \langle\cdot|\cdot\rangle)$ is then called an *inner product space*.

An inner product induces a *norm function* $||\cdot||$ on V with

$$||x|| = \sqrt{\langle x|x\rangle}, \qquad \forall x \in V.$$

We call a vector $x \in V$ *unit* if its squared norm equals one. That is, x is unit if $||x||^2 = (\sqrt{\langle x|x\rangle})^2 = \langle x|x\rangle = 1$.

As is custom in quantum computing, from now on we switch to Dirac notation. That is, for any vector $x \in V$ we write $|x\rangle$ (also called *ket-x*). Additionally, we write $\langle x|$ (also called *bra-x*) for $\langle x|\cdot\rangle$, the inner product function where the left parameter is fixed to x. While $|x\rangle$ is just different notation for a vector x, $\langle x|$ denotes a function $\langle x| : V \to \mathbb{C}$. More precisely, $\langle x|$ is the inner product function where the first parameter is fixed to x.

Let V, W be two complex vector spaces. A function A with $A : V \to W$ is called a *linear operator* if $\forall \lambda_v \in \mathbb{C}$ and $\forall\, |v\rangle \in V$ we have

$$A\Big(\sum_{|v\rangle \in V} \lambda_v |v\rangle\Big) = \Big(\sum_{|v\rangle \in V} \lambda_i A |v\rangle\Big).$$

Linear operators can be viewed as matrices of finite dimensions and applying a linear operator A to some vector $|x\rangle \in V$ corresponds to matrix-vector multiplication. Correspondingly, from now on we write $A|x\rangle$ instead of $A(|x\rangle)$. When we apply m operators $A_1, A_2, \ldots, A_m$ in succession, we write $(A_1 \cdot A_2 \cdots A_n)|x\rangle$. Note that here the $\cdot$ symbol denotes matrix-matrix multiplication.

The *identity* operator I satisfies $I|x\rangle = |x\rangle \ \ \forall |x\rangle \in V$. For any linear operator A, we denote its *inverse* operator as A^{-1}, that is, we have $AA^1 = I$.

For any linear operator A, there exists a unique linear operator $A^\dagger$ such that

$$\langle x|Ay\rangle = \left\langle A^\dagger x \middle| y \right\rangle \qquad \forall x, y \in V.$$

$A^\dagger$ is called the *adjoint* operator of A. From the definition, we can deduce an interesting property of the adjoint operator. We have that

$$\langle x|(AB)y\rangle = \left\langle A^\dagger x \middle| By \right\rangle = \left\langle B^\dagger A^\dagger x \middle| y \right\rangle.$$

Since both x and y are chosen arbitrarily, it holds that $(AB)^\dagger = B^\dagger A^\dagger$.

A linear operator A is called *self-adjoint* if we have $A^\dagger = A$. A linear operator A is called *unitary* if $A^\dagger A = AA^\dagger = I$. We note that for unitary operators, $A^\dagger = A^{-1}$. If an operator A is both unitary and self-adjoint, we have $A^{-1}A = AA^{-1} = AA = I$.

We also require further properties of linear operators that are proven in Chapter 3 of Sheldon Axler's book [7]. The *dimension* of any vector space V, denoted $\dim(V)$, is the size of a basis of V. Let A be a linear operator that maps from some vector space V to another vector space W. Then we define the *kernel* of A to be $\ker(A) = \{|v\rangle \mid |v\rangle \in V, A|v\rangle = |\mathbf{0}\rangle\}$. Furthermore, we define the *range* of A to be the set $\{A|v\rangle \mid |v\rangle \in V\}$. It can be shown that both $\ker(A)$ and $\text{range}(A)$ are vector spaces.

Axler presents a proof that for any linear operator A we have that $\dim(\text{range}(A))$ equals the *rank* of A, which is the number of linearly independent rows (or columns) of A [7]. He goes on to prove what he calls the 'Fundamental Theorem if Linear Maps', which states that for any finite-dimensional vector space V and any linear operator A that maps from V to some other vector space W we have that $\dim(V) = \dim(\ker(A)) + \dim(\text{range}(A))$ [7]. We combine both results into the following variant of the Fundamental Theorem of Linear Maps.

Theorem 1 (Fundamental Theorem of Linear Maps)**.** *Let V be a finite-dimensional vector space and A is a linear operator that maps from V to some other vector space W, we have that*

$$dim(V) = dim(ker(A)) + rank(A).$$

2.3 Quantum-Mechanical Basics

So far, we discussed general mathematical principles. We now apply those principles to quantum computing, loosely following Sections 2.1 and 2.2 in [27] by Nielsen and Chuang. In general, *quantum registers* of size n are mathematically modeled as 2^n-dimensional inner product spaces. We define the following inner product function

$$\langle x|y\rangle = \langle x_{2^n-1}, \dots x_0|y_{2^n-1}, \dots, y_0\rangle = \begin{pmatrix} \overline{x_{2^n-1}} & \dots & \overline{x_0} \end{pmatrix} \cdot \begin{pmatrix} y_{2^n-1} \\ \vdots \\ y_0 \end{pmatrix} \qquad \forall x, y \in \mathbb{C}^{2^n}.$$

Having fixed the inner product function, it is immediately obvious that the computational standard basis is *orthonormal*, that is for arbitrary basis elements $|x\rangle, |y\rangle$ we have that

- $\langle x|x\rangle = 1$ and
- $\langle x|y\rangle = 0 \iff |x\rangle \neq |y\rangle$.

So far we defined adjoint operators for linear operators only. However, in our setting, finding the adjoint of an arbitrary linear operator A corresponds to calculating its complex conjugate transpose. We thus define an *adjoint* value for vectors as well, which we also set to the complex conjugate transpose, that is we define $|x\rangle^\dagger = \langle x|$ for arbitrary $|x\rangle$. We thus get $(A|x\rangle)^\dagger = \langle x|A^\dagger$ and, if A is unitary, we get $(A|x\rangle)^\dagger = \langle x|A^{-1}$.

Another important concept for quantum computing is the *tensor product*. When A is an $n \times m$ matrix and B is a $p \times q$ matrix, then one way to define the tensor product is

$$A \otimes B = \underbrace{\left.\begin{pmatrix} A_{1,1} \cdot B & A_{1,2} \cdot B & \dots & A_{1,m} \cdot B \\ A_{2,1} \cdot B & A_{2,2} \cdot B & \dots & A_{2,m} \cdot B \\ \vdots & \vdots & \vdots & \vdots \\ A_{n,1} \cdot B & A_{n,n} \cdot B & \dots & A_{n,m} \cdot B \end{pmatrix}\right\} n \times p}_{m \times q}$$

Note that here the $\cdot$ symbol denotes scalar-matrix multiplication. For any matrix A, $A^{\otimes 0} = (1)$ where (1) is a one-by-one matrix. Note that for any matrix A we also have $A \otimes (1) = A$. When we have a set of matrices $A = \{A_{(1)}, \dots, A_{(|A|)}\}$, we use the following shorthand for the tensor product over all its elements

$$\bigotimes_{x \in A} x = A_{(1)} \cdots A_{(|A|)}.$$

We first examine important properties of the tensor product when applied to elements of vector spaces. Let V, W denote complex vector spaces and let $|v\rangle, |v'\rangle |w\rangle, |w'\rangle$ denote vectors from V and W respectively and let $\lambda \in \mathbb{C}$. Then the following properties can be deduced for the tensor product.

- $\lambda \cdot (|v\rangle \otimes |w\rangle) = (\lambda \cdot |v\rangle) \otimes |w\rangle = |v\rangle \otimes (\lambda \cdot |w\rangle)$.
- $(|v\rangle + |v'\rangle) \otimes |w\rangle = (|v\rangle \otimes |w\rangle) + (|v'\rangle \otimes |w\rangle)$.
- $|v\rangle \otimes (|w\rangle + |w'\rangle) = (|v\rangle \otimes |w\rangle) + (|v\rangle \otimes |w'\rangle)$.

Above we defined the tensor product as a function over matrices, so we can also apply the tensor product to the matrix representations of linear operators. Let A, B be two linear operators (represented as matrices) on some complex vector spaces V, W and let $|v\rangle \in V$ and $|w\rangle \in W$. Then an important property used constantly in quantum computing is

$$(A \otimes B)(|v\rangle \otimes |w\rangle) = (A|v\rangle \otimes B|w\rangle).$$

A single *qubit* is a unit vector from $\mathbb{C}^{2^1} = \mathbb{C}^2$. That is, a single qubit $|q\rangle$ has the form $|q\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$ with $\alpha, \beta \in \mathbb{C}$ and $\langle q|q\rangle = \overline{\alpha}\alpha + \overline{\beta}\beta = 1$. Following convention, we introduce dedicated symbols for the *basis states* of $\mathbb{C}^2$, we define

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle\,, \text{ and } \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle\,.$$

Moreover, we will always write non-basis states as decompositions of basis states. That is, instead of writing $\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$, we write $\alpha \cdot |0\rangle + \beta \cdot |1\rangle$. A non-basis state like this is called a *superposition* and α, β are called the *amplitudes* of the respective basis states.

Multi-qubit registers are composed of single qubits via the tensor product. We concatenate two qubits $|q_1\rangle, |q_0\rangle$ by calculating $|q_1\rangle \otimes |q_0\rangle$, and usually we omit the $\otimes$ symbol and just write $|q_1\rangle\,|q_0\rangle$ or $|q_1 q_0\rangle$ (which is now a unit vector in $\mathbb{C}^4$). Let x denote a bitstring of length n. Then we write $|x_{n-1}\rangle \otimes |x_{n-2}\rangle \otimes \cdots \otimes |x_0\rangle = |x_{n-1}x_{n-2}\ldots x_0\rangle = |x\rangle$, where x_0 is the least significant bit and x_{n-1} is the most significant bit. We say that the quantum register $|x\rangle$ *holds* the bitstring x.

Note also that the set $\mathfrak{B}_n = \{|x\rangle \mid x \in \{0,1\}^n\}$ forms the computational standard basis for the vector space $\mathbb{C}^{2^n}$. Recall that the computational standard basis (with respect to our fixed inner product function) is orthogonal. Hence, for any two basis elements $|x\rangle, |y\rangle \in \mathfrak{B}_n$ with $|x\rangle \neq |y\rangle$ we have that $\langle x|y\rangle = 0$.

Quantum registers are manipulated by unitary operators. If A is a unitary operator with $A : \mathbb{C}^2 \to \mathbb{C}^2$ and $|q\rangle$ is a qubit, then we apply A to $|q\rangle$ by calculating $A \cdot |q\rangle$, where $\cdot$ denotes matrix-vector multiplication. Similarly to how we construct larger quantum registers from smaller ones, we can construct larger quantum operators from smaller ones via the tensor product. If we want to apply A to $|q_1 q_0\rangle$, we construct a new operator $A \otimes A$ and calculate $(A \otimes A)\,|q_1 q_0\rangle$. If we want to apply A to each qubit of an n qubit register $|x\rangle$, we construct $A^{\otimes n} = \underbrace{A \otimes A \otimes \cdots \otimes A}_{n \text{ times}}$ and calculate $A^{\otimes n}\,|x\rangle$.

Let $|x\rangle$ be a quantum register of length $n + m$. Let A, B be unitary operators with $A : \mathbb{C}^{2^n} \to \mathbb{C}^{2^n}$ and $B : \mathbb{C}^{2^m} \to \mathbb{C}^{2^m}$. Then we can simultaneously apply A to the first n qubits and B to the last m qubits of $|x\rangle$ by constructing the operator $A \otimes B$ and calculating $(A \otimes B)\,|x\rangle$. We call the successive application of unitary operators to a quantum register a *quantum algorithm*.

The operation of *measuring* a quantum gate corresponds to projecting it to a basis state. Consider a quantum register in the following superposition

$$|x\rangle = \sum_{y \in X \subseteq \{0,1\}^n} \lambda_y\,|y\rangle\,.$$

Then the probability to measure any particular y is given as

$$\langle \lambda_y y|\lambda_y y\rangle = \lambda_y \overline{\lambda_y}\,\langle y|y\rangle = (||\lambda_y||)^2.$$

We use $|\psi\rangle$ or $|\Psi\rangle$ when we want to assign names to quantum states. Furthermore, following Brassard and Høyer [13], we introduce shorthands for certain recurring superposition states. Let $G = \langle \{0,1\}^n, \oplus \rangle$, then for any non-empty $X \subseteq G$ and $g \in G$ we use the shorthands

$$|g \oplus X\rangle = \frac{1}{\sqrt{|X|}} \sum_{x \in X} |g \oplus x\rangle \,, \qquad |\phi_g X\rangle = \frac{1}{\sqrt{|X|}} \sum_{x \in X} (-1)^{g \odot x} |x\rangle \,.$$

2.4 Operators in Quantum Computing

We give an overview over the standard quantum operators that are used in the algorithms presented in this thesis, starting with those operators that act on a single qubit, the symbol $\mathcal{I}$ denotes the quantum mechanical *identity* operator. Furthermore, $\mathcal{X}$ denotes the *Pauli X* operator and $\mathcal{H}$ denotes the *Hadamard* operator. They are commonly defined as

$$\mathcal{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \mathcal{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad \mathcal{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

The operator $\mathcal{I}$ was already covered before in the section on linear algebra basics. The operator $\mathcal{X}$ has a nice interpretation as the quantum analog to the logical negation, we have $\mathcal{X}|0\rangle = |1\rangle$ and $\mathcal{X}|1\rangle = |0\rangle$.

The Hadamard operator is a crucial building block for most quantum algorithms as it can be used to create superposition states. Its effects are

$$\mathcal{H}|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \text{ and } \mathcal{H}|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

Let $|x\rangle$ be some n-qubit quantum state. Then another important property of the Hadamard operator [13], which we will make use of frequently is

$$\mathcal{H}^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{y \odot x} |y\rangle \,.$$

The gates mentioned so far are all self-adjoint, hence we have $\mathcal{I}\mathcal{I} = \mathcal{X}\mathcal{X} = \mathcal{H}\mathcal{H} = \mathcal{I}$.

The symbol $\mathcal{S}$ denotes the *phase* operator

$$\mathcal{S} = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}.$$

We note that $\mathcal{S}|0\rangle = |0\rangle$, that is, the operator $\mathcal{S}$ only has effects when applied to $|1\rangle$. Importantly, the $\mathcal{S}$ operator is not self-adjoint. On paper, we can easily construct $\mathcal{S}^{-1}$ by just taking the complex conjugate transpose of the matrix of $\mathcal{S}$. A more implementation-oriented approach relying on operators already introduced would be just to repeat applications $\mathcal{S}$, we have $(\mathcal{S}\mathcal{S}\mathcal{S})\mathcal{S}|q\rangle = |q\rangle \,, |q\rangle \in \{|0\rangle \,, |1\rangle\}$.

We turn our attention towards operators on multiple qubits. Sometimes, we want to apply an operator conditionally, depending on the current state of a register. This is where *controlled* operations come in. The two-qubit $\mathcal{CNOT}$ operator (also *controlled* $\mathcal{X}$ operator) is defined as

$$\mathcal{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

It never changes the first qubit, also called the *control* qubit. If the control qubit is in state $|1\rangle$, the second qubit, called the *target* qubit, is negated. That is we have $\mathcal{CNOT}\,|q_1\rangle\,|q_0\rangle = |q_1\rangle\,|q_1 \oplus q_0\rangle$.

The three-qubit *Toffoli* operator, also called $\mathcal{CCNOT}$, is similar to the regular $\mathcal{CNOT}$ operator, but this time we have two control qubits. The $\mathcal{CCNOT}$ operator is defined as

$$\mathcal{CCNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

In other words, we have $\mathcal{CCNOT}\,|11\rangle\,|q_0\rangle = |11\rangle\,\mathcal{X}\,|q_0\rangle$ and $\mathcal{CCNOT}\,|x_1x_0\rangle\,|q_0\rangle = |x_1x_0\rangle\,|q_0\rangle$ when $|x_1\rangle, |x_0\rangle$ are not both $|1\rangle$.

We can also *control* other one-qubit operators. If A is a quantum operator, then its controlled version is denoted $A_{(q_{n-1}\wedge\cdots\wedge q_0),q_n}$. The semantics of this notation is that we apply A to $|q_n\rangle$ if and only if $|q_{n-1}\rangle = \cdots = |q_0\rangle = |1\rangle$. Arbitrary controlled operators are non-standard and have to be simulated individually.

It is easy to see that the $\mathcal{CNOT}$ and $\mathcal{CCNOT}$ operators are self-adjoint. For arbitrary controlled operators, the self-adjoint property depends on the concrete operator that is being controlled.

Let f be a function with $f : \{0,1\}^n \to \{0,1\}^m$. Then we denote the operator that implements f as $\mathcal{U}_f$, and we have $\mathcal{U}_f\,|x\rangle\,|y\rangle = |x\rangle\,|y \oplus f(y)\rangle$, where $|x\rangle$ is an n-qubit register and $|y\rangle$ is an m-qubit register. The matrix representation of $\mathcal{U}_f$ depends on the function f, thus it needs to be stated individually for each f. However, for arbitrary f, the $\mathcal{U}_f$ operator is self-adjoint.

CHAPTER 3

Classical and Quantum Complexity

In this section we give an overview over the classical and quantum complexity classes relevant for this thesis. We recapitulate the circuit model of computation which we will use later for analyzing the complexity of the quantum algorithms presented here. We also formally introduce the complexity measure *oracle complexity* which is highly relevant for Simon's problem and last we discuss challenges for quantum computation in practice.

3.1 Classical Complexity Theory

The complexity class $\mathcal{P}$ (for *polynomial time*) describes all problems whose instances can be solved by a deterministic Turing machine in time polynomial in the instance size. More formally, let $\mathbf{P}$ be a problem and let p be an arbitrary instance of $\mathbf{P}$. Then $\mathbf{P}$ is contained in $\mathcal{P}$ precisely if there exists some polynomial function f such that a deterministic Turing machine halts on p after $f(|p|)$ steps (applications of the transition function) where $|p|$ denotes the size of p. Problems in $\mathcal{P}$ are considered tractable.

The complexity class $\mathcal{BPP}$ (for *bounded-error probabilistic polynomial time*) describes all problems whose instances can be solved by a probabilistic Turing machine with bounded error probability in polynomial time. Note that some authors give a concrete number for the error probability bound when defining $\mathcal{BPP}$, for example $\frac{1}{3}$ or $\frac{1}{4}$. As long as the upper bound is constant, the concrete number essentially does not matter. This is because we can boost the success probability of the probabilistic Turing machine by running it repeatedly on the same input. Problems in $\mathcal{BPP}$ are still considered tractable [27].

3.2 The Circuit Model of Computation

Turing machines help us to reason about a problem on an abstract level. However, apart from introductory courses to computer science, we never actually specify, let alone physically construct Turing machines when we implement an algorithm for a problem. Instead, we write a computer program in a programming language of our choice, which then gets compiled, ultimately, into a Boolean logic circuit. Intuitively, the circuit consists of *wires* that connect *bits* (as many as our CPU has available) to *logic gates* (those gates that the CPU implements). We then run the circuit on our CPU and obtain our result. This process has been formalized and is known as the *circuit model of computation* and in its presentation we follow Nielsen and Chuang [27].

With circuits, we aim to model actual hardware. Contrary to Turing machines which have infinitely long tapes, actual computing hardware is finite, and thus we want to reason about finite circuits. In the Turing machine model, we have only one Turing machine that solves arbitrary instances of a problem, but in the circuit model, different instances (with respect to size) require different circuits. Thus, a problem is not solved by any one circuit, but by a set (often called *family*) of circuits often denoted $\{C_n\}, n \in \mathbb{N}^+$. Intuitively, C_n stands for the circuit that solves problem instances of size n. A circuit family $\{C_n\}$ is called *uniform* if for any problem instance p of some problem $\mathbf{P}$ there exists a Turing machine that 'efficiently' constructs C_n.

For a uniform circuit family, the size of a generated circuit relative to its input coincides with the notion of complexity we have in the Turing machine model. For example, if there is an exact uniform circuit family for $\mathbf{P}$ where each circuit has size polynomial in its input, then $\mathbf{P} \in \mathcal{P}$. Popular measures for the size of a circuit are first its width, which is the number of bits acted upon. Circuit width corresponds to space complexity from the Turing machine model. A second popular circuit size measure is circuit depth, where we count the number of gates that are executed upon each individual bit, and then we take the maximum result. Circuit depth corresponds to time complexity in the Turing machine model, and we will use circuit depth to describe the complexity of the algorithms presented in this thesis.

3.3 Quantum Complexity Theory

Similar to the Turing machine which is used as a formalism to decide complexity and computability on classical hardware, there is the concept of a *quantum Turing machine* for quantum hardware [11]. We omit the details but note one key difference to classical Turing machines. Classical Turing machines have a tape which we can read at any time and as often as we want. Quantum Turing machines have a tape that we can only ever read once, usually at the very end of some computation. This process of reading corresponds to the measurement operation defined in Section 2.3

Similar to the classical complexity theory, we classify problems by how long it takes a quantum Turing machine to solve them. The quantum complexity class analog to $\mathcal{P}$ is

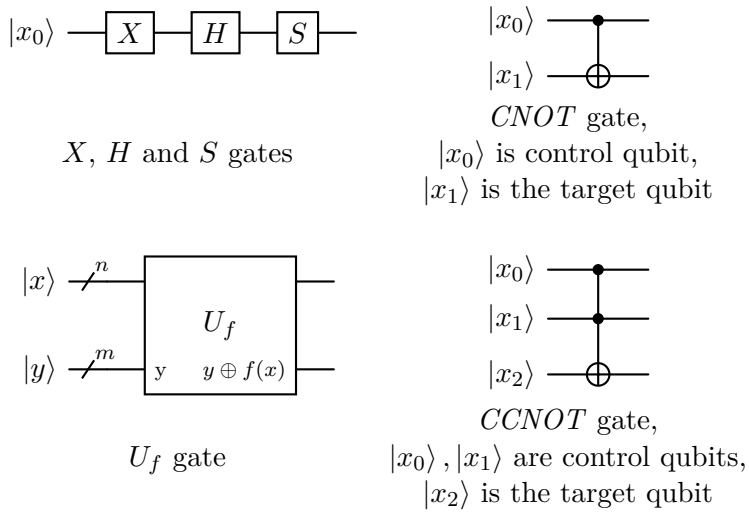
called $\mathcal{EQP}$ (for *exact quantum polynomial time*). It contains all problems that can be solved by a quantum Turing machine in polynomial time. More formally, a problem **P** is contained in $\mathcal{EQP}$ if there exist a quantum Turing machine with a dedicated *acceptance* cell and a polynomial function f such that for every instance p of **P**, we are guaranteed to measure that acceptance cell after $f(|p|)$ time steps [11].

The class $\mathcal{BQP}$ (for *bounded-error quantum polynomial*) is the quantum analog to the classical complexity class $\mathcal{BPP}$. As described above for $\mathcal{EQP}$, it contains all problems that can be solved by a quantum Turing machine in polynomial time, but now we drop the restriction that we must always measure the acceptance cell in the end [11]. We again admit a bounded error probability.

3.4 Quantum Circuits

As with classical computing, as an alternative to Turing machines, we can use the circuit model of computation for quantum computers. The key difference is that we now do not construct Boolean logic circuits, but *quantum circuits*. Quantum circuits are made from qubits and *quantum gates*, connected by *quantum wires*. They are not executed on a regular CPU but on a quantum processing unit (QPU) [27].

The quantum gates for a particular QPU are the quantum operators that the QPU supports. When we use the operators described in Section 2.4 as gates in a quantum circuit, we denote them like shown in Figure 3.1. We use the same names for quantum operators and quantum gates. When we reason about operators, we use calligraphic letters and when we reason about gates, we use standard Latin letters. In our figures depicting quantum circuits, we always use Qiskit bit order. That is, the least significant qubit is topmost.



X, H and S gates

CNOT gate,
$|x_0\rangle$ is control qubit,
$|x_1\rangle$ is the target qubit

U_f gate

CCNOT gate,
$|x_0\rangle$, $|x_1\rangle$ are control qubits,
$|x_2\rangle$ is the target qubit

Figure 3.1: Quantum Gates

As with the classical circuit model of computation, for a problem to be placed in a certain complexity class, we need a corresponding uniform circuit family where the size of the generated circuits meets some criteria based on the problem input size. All quantum algorithms presented in this thesis are given as quantum circuits, along with a uniform specification on how to generate them. Hence, the complexity of the algorithms in the quantum Turing machine sense coincides with the corresponding circuit depths.

3.5 Oracle Complexity

So far we discussed general complexity measures for arbitrary problems. We now introduce a complexity measure for a particular class of problems called *oracle problems*. The input to an oracle problem is a black box implementation of some function ρ, denoted $\mathcal{U}_\rho$. We assume that the output of ρ is not random, but follows a certain pattern. In the literature, ρ is often described as 'fulfilling a promise'. When we are given $\mathcal{U}_\rho$, our task is to reconstruct its pattern. Well known oracle problems are the problem of Deutsch [17], the problem of Bernstein and Vazirani [11] or the problem of Simon [32].

In an oracle problem setting, analyzing the circuit size gives us only incomplete information about the problem's complexity. To illustrate this, let $\mathcal{U}_\rho$ be an instance of an oracle problem. Assume further that we have a circuit in which we repeatedly query $\mathcal{U}_\rho$ with different inputs to reconstruct its pattern. Then, the size of our circuit depends on the circuit size of $\mathcal{U}_\rho$. We cannot control $\mathcal{U}_\rho$ since it is given to us as problem input, and because $\mathcal{U}_\rho$ is given to us as a black box, we cannot even analyze its size. As a consequence, we cannot make adequate statements about the size of our own circuit anymore.

In such a situation it is common to reason about the number of calls to $\mathcal{U}_\rho$ only, that number is widely known as oracle complexity. This measure is particularly useful when a call to $\mathcal{U}_\rho$ is expensive, i.e. when its circuit depth is large. On the other hand, it is imprecise for situations where calls to $\mathcal{U}_\rho$ are cheap or when we have only few (e.g. linearly many) oracle calls, but then lots of pre- and post-processing (e.g. exponentially many) steps. For a more nuanced approach, we follow Berthiaume and Brassard [12] and treat $\mathcal{U}_\rho$ as an elementary (quantum) Turing machine operation or, in the circuit model, as an elementary (quantum) gate. Then the complexity class $\mathcal{P}^{\mathcal{U}_\rho}$ contains all problems that can be solved by a deterministic Turing machine in polynomial time that can access $\mathcal{U}_\rho$ in constant time. All other complexity classes discussed so far can be augmented with an oracle analogously.

3.6 Quantum Computing in Practice

Quantum computing is in the so-called NISQ (for *Noisy Intermediate-Scale Quantum*) era [29]. Intermediate-scale means that currently only few quantum hardware resources are available. Noisy means that the quantum hardware we do have is very susceptible to faults and imprecision. If we prepare a qubit to a certain state, then with time, it

will evolve to a different state on its own because we currently cannot fully isolate a qubit from its environment. This phenomenon is called *decoherence*. Moreover, if we apply a quantum gate to a qubit, then the resulting state of the qubit might very well deviate substantially from what we would expect from the gate's specification. This happens because current quantum gate implementations are often not capable to perform fine-grained qubit state manipulations with mathematical precision. This phenomenon is widely known as *gate infidelity*. Decoherence and gate infidelity drastically limit the depth up to which circuits can be run with reasonable error probability [22].

Due to NISQ limitations, it is even difficult to just predict, whether a certain quantum computer can successfully run a certain quantum circuit. One widely adopted benchmark measure for quantum processors is *quantum volume (QV)* [16]. Intuitively, the quantum volume score of a quantum processor is the size of the largest *quadratic* quantum circuit (that is a circuit with equal width and depth) from a certain circuit family that can successfully be run on it. While giving a good first estimate of what a processor can do, the quantum volume score is a bad predictor for the performance of circuits that are not quadratic, i.e. where one 'side' is way longer than the other [23]. Additionally, the circuits that are used for the benchmark need to be compiled for the hardware they are executed on, and thus the quality of the compiler in use also impacts the quantum volume score of a given device. As a consequence, for developers without access to sophisticated proprietary compilers, the quantum volume score is an even worse indicator of how well a given circuit might run on any particular quantum processor [28].

CHAPTER 4

A Randomized Algorithm for Simon's Problem

We now give a formal definition of Simon's problem, sticking closely to the original presentation [32].

- **Input**: A black box implementation for some function $\rho : \{0,1\}^n \to \{0,1\}^m$ with $m \geq n$. Furthermore, ρ fulfills the promise that either
  - ρ is bijective or
  - there exists a non-trivial bitstring s such that for all $g, g' \in \{0,1\}^n$, $g \neq g'$, we have

$$\rho(g) = \rho(g') \iff g' = g \oplus s.$$

- **Output**: In case ρ is bijective, a random string. Otherwise s.

In a classical setting, Simon's problem is hard. The best known algorithms have exponential oracle complexity, both in the deterministic and in the randomized case [32]. As a consequence, solving Simon's problem using classical hardware only is likely infeasible even for problem instances where the implementation of ρ has constant runtime.

As already mentioned in the introduction, Daniel Simon also proposed what is now known as Simon's algorithm. It is a mixed quantum-classical algorithm that solves Simon's problem with only linear oracle complexity. That means, when we use oracle complexity as complexity measure, Simon's hybrid algorithm has exponential speedup over all known purely classical ones. In the algorithm, we first run a fixed quantum circuit multiple times, and each such run yields a bitstring $h \in \{0,1\}^n$. Once we have collected 'enough' of them, we use a classical computer to form a system of linear equations. If ρ is bijective,

then the solution to that system of equations is some random bitstring. In the other case however, the solution is the secret string s.

In this chapter we first give a formal description and mathematical analysis of the quantum algorithm proposed by Simon [32]. We prove its correctness and we formalize what it means to have 'enough' bitstrings. Next, we analyze the classical post-processing step, and finally we present an example run of the entire algorithm.

4.1 Quantum Algorithm for Simon's Problem

Before we give a formal description of Simon's algorithm, we state a purely technical lemma which will help us simplify the upcoming mathematical analysis.

Lemma 1 (Distributivity Lemma)**.** *Let g, k, y be three arbitrary bitstrings of equal length. Then we have*

$$(-1)^{g\odot(y\oplus k)} = (-1)^{g\odot y}(-1)^{g\odot k}.$$

Proof. First note that for an arbitrary index i, $0 \leq i < n$ and arbitrary bitstrings g, y, k of length n we have

$$(g_i \cdot y_i) \oplus (g_i \cdot k_i) = g_i \cdot (y_i \oplus k_i).$$

This can be easily verified by performing a case distinction on y and k. If $y_i = k_i$, then we have

$$g_i \cdot y_i \oplus g_i \cdot k_i = 0 = g_i \cdot 0 = g_i \cdot (y_i \oplus k_i).$$

If on the other hand we have $y_i \neq k_i$, then we have

$$g_i \cdot y_i \oplus g_i \cdot k_i = g_i = g_i \cdot 1 = g_i \cdot (y_i \oplus k_i).$$

Now we can write

$$\begin{aligned} g \odot (y \oplus k) &= g_{n-1} \cdot (y_{n-1} \oplus k_{n-1}) \oplus \cdots \oplus g_0 \cdot (y_0 \oplus k_0) \\ &= (g_{n-1} \cdot y_{n-1}) \oplus (g_{n-1} \cdot k_{n-1}) \oplus \cdots \oplus (g_0 \cdot y_0) \oplus (g_0 \cdot k_0) \\ &= (g \odot y) \oplus (g \odot k). \end{aligned}$$

Hence, we have

$$(-1)^{g\odot(y\oplus k)} = (-1)^{(g\odot y)\oplus(g\odot k)} = (-1)^{(g\odot y)+(g\odot k)} = (-1)^{g\odot y}(-1)^{g\odot k}. \qquad \square$$

The quantum part of Simon's algorithm can be defined as in Algorithm 4.1. It operates on two quantum registers, one of size n and the other of size m. Furthermore, we assume that we are given a black box quantum implementation of ρ in the form of the quantum operator $\mathcal{U}_\rho$, for which we have $\mathcal{U}_\rho |x\rangle |y\rangle = |x\rangle |y \oplus \rho(x)\rangle$. The presentation in Algorithm

4.1 stays close to the original one given by Simon [32], we only deviate in step 4 (state $|\psi_2'\rangle$), which is not part of the original presentation. This extra step is not strictly necessary, but it drastically simplifies the mathematical analysis of the algorithm, and it is included in many corresponding textbooks and articles [20, 3].

Algorithm 4.1: Quantum part of Simon's Algorithm, simplified

1. $|\psi_0\rangle = |0\rangle^{\otimes n} |0\rangle^{\otimes m}$ — initial state
2. $|\psi_1\rangle = (\mathcal{H}^{\otimes n} \otimes \mathcal{I}^{\otimes m}) |\psi_0\rangle$ — apply $\mathcal{H}$ to each qubit in first register
3. $|\psi_2\rangle = \mathcal{U}_\rho |\psi_1\rangle$ — apply oracle for ρ
4. $|\psi_2'\rangle$ — measure the second register
5. $|\psi_3\rangle = (\mathcal{H}^{\otimes n} \otimes \mathcal{I}^{\otimes m}) |\psi_2\rangle$ — apply $\mathcal{H}$ to each qubit in first register
6. Obtain h by measuring the first register.

In the next theorem, we perform the mathematical analysis of Simon's algorithm as described in Algorithm 4.1.

Theorem 2. *Let ρ be an instance of Simon's problem. Then Simon's quantum algorithm returns a random element $h \in \{0,1\}^n$ if ρ is bijective. Otherwise, the algorithm returns an element $h \in \{0,1\}^n$ with $s \odot h = 0$ where s is the secret string of ρ. Moreover, Simon's quantum algorithm can be implemented as a quantum circuit with the same depth as the circuit for $\mathcal{U}_\rho$ plus a constant.*

Proof. We first analyze the states of Simon's algorithm (Algorithm 4.1) in more detail, starting with $|\psi_1\rangle$. We have

$$|\psi_1\rangle = (\mathcal{H}^{\otimes n} \otimes \mathcal{I}^{\otimes m}) |\psi_0\rangle = \mathcal{H}^{\otimes n} |0\rangle^{\otimes n} |0\rangle^{\otimes m} = \frac{1}{\sqrt{2^n}} \Big(\sum_{g \in \{0,1\}^n} (-1)^{g \odot \mathbf{0}} |g\rangle \Big) |0\rangle^{\otimes m}$$
$$= \frac{1}{\sqrt{2^n}} \Big(\sum_{g \in \{0,1\}^n} |g\rangle \Big) |0\rangle^{\otimes m}.$$

Next we apply $\mathcal{U}_\rho$ to $|\psi_1\rangle$, resulting in

$$|\psi_2\rangle = \mathcal{U}_\rho |\psi_1\rangle = \frac{1}{\sqrt{2^n}} \Big(\sum_{g \in \{0,1\}^n} |g\rangle |\mathbf{0} \oplus \rho(g)\rangle \Big) = \frac{1}{\sqrt{2^n}} \Big(\sum_{g \in \{0,1\}^n} |g\rangle |\rho(g)\rangle \Big).$$

In the next step, we measure the second register, thus fixing one particular $\rho(g)$ for some $g \in \{0,1\}^n$ in the second register. Since we applied $\mathcal{U}_\rho$ in the previous step, both registers are entangled and fixing $\rho(g)$ also influences the superposition in the first register. The value held in the first register now depends on whether ρ is bijective or there exists a

non-trivial bitstring s such that $\rho(g) = \rho(g') \iff g' = g \oplus s$ (where $g' \in \{0,1\}^n, g \neq g'$). In the first case, we have $|\psi_2'\rangle = |g\rangle$ for some $g \in \{0,1\}^n$. In the second case we have $|\psi_2'\rangle = \frac{1}{\sqrt{2}}(|g\rangle + |g \oplus s\rangle)$.

We proceed with the next step. Note that in our description of $|\psi_2'\rangle$, we already omitted the second register, because we measured it and we will not use it again. From now on, we are working with the first quantum register of size n only. In case ρ is bijective, we have

$$|\psi_3\rangle = \mathcal{H}^{\otimes n}|\psi_2'\rangle = \mathcal{H}^{\otimes n}|g\rangle = \frac{1}{\sqrt{2^n}}\Big(\sum_{h\in\{0,1\}^n}(-1)^{g\odot h}|h\rangle\Big).$$

Hence, when we measure $|\psi_3\rangle$, we will obtain some random bitstring in $\{0,1\}^n$. In the other case, the analysis is more involved. We start with

$$\begin{aligned}
|\psi_3\rangle = \mathcal{H}^{\otimes n}|\psi_2'\rangle &= \mathcal{H}^{\otimes n}\frac{1}{\sqrt{2}}\Big(|g\rangle + |g\oplus s\rangle\Big) = \frac{1}{\sqrt{2}}\Big(\mathcal{H}^{\otimes n}|g\rangle + \mathcal{H}^{\otimes n}|g\oplus s\rangle\Big)\\
&= \frac{1}{\sqrt{2}}\Big(\frac{1}{\sqrt{2^n}}\Big(\sum_{h\in\{0,1\}^n}(-1)^{g\odot h}|h\rangle\Big) + \frac{1}{\sqrt{2^n}}\Big(\sum_{h\in\{0,1\}^n}(-1)^{(g\oplus s)\odot h}|h\rangle\Big)\Big)\\
&= \frac{1}{\sqrt{2}}\frac{1}{\sqrt{2^n}}\Big(\sum_{h\in\{0,1\}^n}(-1)^{g\odot h}|h\rangle + \sum_{h\in\{0,1\}^n}(-1)^{(g\oplus s)\odot h}|h\rangle\Big)\\
&= \frac{1}{\sqrt{2^{n+1}}}\Big(\sum_{h\in\{0,1\}^n}(-1)^{g\odot h}|h\rangle + (-1)^{(g\oplus s)\odot h}|h\rangle\Big)\\
&= \frac{1}{\sqrt{2^{n+1}}}\Big(\sum_{h\in\{0,1\}^n}((-1)^{g\odot h} + (-1)^{(g\oplus s)\odot h})|h\rangle\Big).
\end{aligned}$$

By the Distributivity Lemma (Lemma 1), we can rewrite $|\psi_3\rangle$ further into

$$\begin{aligned}
|\psi_3\rangle &= \frac{1}{\sqrt{2^{n+1}}}\Big(\sum_{h\in\{0,1\}^n}((-1)^{g\odot h} + (-1)^{g\odot h}(-1)^{(s\odot h)})|h\rangle\Big)\\
&= \frac{1}{\sqrt{2^{n+1}}}\Big(\sum_{h\in\{0,1\}^n}(-1)^{g\odot h}(1 + (-1)^{(s\odot h)})|h\rangle\Big).
\end{aligned}$$

We now fix an arbitrary $h \in \{0,1\}^n$ with $s \odot h = 1$, and we analyze its amplitude in $|\psi_3\rangle$. It is given as

$$\frac{1}{\sqrt{2^{n+1}}}\Big((-1)^{g\odot h}(1 + (-1)^{(s\odot h)})\Big) = \frac{1}{\sqrt{2^{n+1}}}\Big((-1)^{g\odot h}(1 + (-1))\Big) = 0.$$

Hence, when we measure the register in $|\psi_3\rangle$, we are guaranteed to measure a bitstring h with $s \odot h = 0$.

Step 2. clearly has circuit depth 1. Step 3. has precisely the circuit depth of the circuit of $\mathcal{U}_\rho$. The steps 4. and 5. have circuit depth 1 again, hence the total depth of the circuit is the depth of the circuit of $\mathcal{U}_\rho + 3$. □

The quantum part of Simon's algorithm consists of repeatedly running Algorithm 4.1 until we have collected precisely $n-1$ linearly independent bitstrings from $\{0,1\}^n$. Each time, we run the same quantum circuit, so it might happen that we repeatedly measure bitstrings that are linearly dependent from those we already measured previously. Hence, Simon's algorithm is not deterministic, but randomized, and we cannot give a fixed upper runtime bound, only an expected one. Later in Section 4.4 we are going to prove that after $O(n)$ repetitions of Algorithm 4.1 we will have measured $n-1$ linearly independent bitstrings with very high probability.

4.2 Classical Post-Processing for Simon's Problem

Assume that for an instance $\mathcal{U}_\rho$ of Simon's problem, we are given a linearly independent set of bitstrings $\{h^{(1)}, \dots, h^{(n-1)}\}$ obtained by repeatedly running Simon's quantum algorithm (Algorithm 4.1). We now use those bitstrings to construct a system of linear equations as follows

$$\begin{aligned} h^{(1)}_{n-1}x_{n-1} + h^{(1)}_{n-2}x_{n-2} + \cdots + h^{(1)}_0 x_0 &\equiv 0 \pmod 2 \\ &\dots \\ h^{(n-1)}_{n-1}x_{n-1} + h^{(n-1)}_{n-2}x_{n-2} + \cdots + h^{(n-1)}_0 x_0 &\equiv 0 \pmod 2. \end{aligned}$$

The above system has the trivial solution $x = \mathbf{0}$. Since we are working with equations with n unknowns and since we have $n-1$ linearly independent equations, the system also has one unique non-trivial solution. We can obtain that solution (on classical hardware) via Gaussian elimination. We now perform a case distinction.

First, we consider the case that ρ is bijective. Then the nontrivial solution x to our system of equations is some random bitstring. Consider now the case that ρ is not bijective. Then there exists a secret string s such that for all $g, g' \in \{0,1\}^n, g \neq g'$ we have that $\rho(g) = \rho(g') \iff g' = g \oplus s$. By Theorem 2, for $1 \leq i \leq n-1$ we have $h^{(i)} \odot s = 0$, which is equivalent to the condition

$$h^{(i)}_{n-1}s_{n-1} + h^{(i)}_{n-2}s_{n-2} + \cdots + h^{(i)}_0 s_0 \equiv 0 \pmod 2.$$

Hence, the non-trivial solution to our system of equations is s itself.

Given a non-trivial solution s' to the system of equations above, we still need to distinguish between the cases of ρ being bijective or not. We do this by running both $\rho(\mathbf{0})$ and $\rho(s')$ and comparing the results.

Lemma 2. *If $\rho(s') \neq \rho(\mathbf{0})$, then ρ must be bijective.*

Proof. Assume towards a contradiction that $\rho(s') \neq \rho(\mathbf{0})$, but ρ is not bijective. We recall from above that if ρ is not bijective, then $s' = s$ and for all $g, g' \in \{0,1\}^n$ we have $\rho(g) = \rho(g') \iff g' = g \oplus s$. In particular, we then must have $\rho(\mathbf{0}) = \rho(s') \iff s' = \mathbf{0} \oplus s$. Since $s' = s = \mathbf{0} \oplus s$, we must also have $\rho(s') = \rho(\mathbf{0})$, a contradiction to the original assumption. □

Lemma 3. *If $\rho(s') = \rho(\mathbf{0})$, then ρ is not bijective.*

Proof. Assume towards a contradiction that $\rho(s') = \rho(\mathbf{0})$, but ρ is bijective. By construction, $s' \neq \mathbf{0}$ and since ρ is bijective, we cannot have $\rho(s') = \rho(\mathbf{0})$. □

4.3 An Example Run of the Original Version of Simon's Algorithm

Let ρ with $\rho : \{0,1\}^3 \to \{0,1\}^2$ be a function fulfilling Simon's promise. Let $s = 001$ and for all $g, g' \in \{0,1\}^3 : \rho(g) = \rho(g') \iff g' = g \oplus s$. A possible implementation $\mathcal{U}_\rho$ is the quantum operator specified as follows

$$\begin{aligned}
&\mathcal{U}_\rho |000\rangle |q_1q_0\rangle \to |000\rangle |q_1q_0 \oplus 00\rangle , && \mathcal{U}_\rho |001\rangle |q_1q_0\rangle \to |001\rangle |q_1q_0 \oplus 00\rangle ,\\
&\mathcal{U}_\rho |010\rangle |q_1q_0\rangle \to |010\rangle |q_1q_0 \oplus 01\rangle , && \mathcal{U}_\rho |011\rangle |q_1q_0\rangle \to |011\rangle |q_1q_0 \oplus 01\rangle ,\\
&\mathcal{U}_\rho |100\rangle |q_1q_0\rangle \to |100\rangle |q_1q_0 \oplus 10\rangle , && \mathcal{U}_\rho |101\rangle |q_1q_0\rangle \to |101\rangle |q_1q_0 \oplus 10\rangle ,\\
&\mathcal{U}_\rho |110\rangle |q_1q_0\rangle \to |110\rangle |q_1q_0 \oplus 11\rangle , && \mathcal{U}_\rho |111\rangle |q_1q_0\rangle \to |111\rangle |q_1q_0 \oplus 11\rangle .
\end{aligned}$$

We run Algorithm 4.1 in a loop. In each loop iteration we have $|\psi_0\rangle = |0\rangle^{\otimes 3} |0\rangle^{\otimes 2}$. Next, we apply the Hadamard operator to the first register and get

$$|\psi_1\rangle = \frac{1}{\sqrt{8}} \Big(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle \Big) |00\rangle .$$

Next, we apply $\mathcal{U}_\rho$, which puts our quantum computer in the state

$$\begin{aligned}
|\psi_2\rangle = \frac{1}{\sqrt{8}} \Big(&|000\rangle |00\rangle + |001\rangle |00\rangle + |010\rangle |01\rangle + |011\rangle |01\rangle \\
&+ |100\rangle |10\rangle + |101\rangle |10\rangle + |110\rangle |11\rangle + |111\rangle |11\rangle \Big).
\end{aligned}$$

We then measure the second register. Assume that we measure the bitstring 01. Then the first register of our quantum computer is in state $|\psi_2'\rangle = \frac{1}{\sqrt{2}}(|010\rangle + |011\rangle)$. We will again omit the second register from future calculations, as we are not going to need it anymore. The next step is to apply the Hadamard operator to each qubit in the first

register, which gives

$$
\begin{aligned}
|\psi_3\rangle = \mathcal{H}^{\otimes 3}|\psi_2'\rangle &= \mathcal{H}^{\otimes 3}\frac{1}{\sqrt{2}}\Big(|010\rangle + |011\rangle\Big) = \frac{1}{\sqrt{2}}\Big(\mathcal{H}^{\otimes 3}|010\rangle + \mathcal{H}^{\otimes 3}|011\rangle\Big) \\
&= \frac{1}{\sqrt{2}}\Big(\frac{1}{\sqrt{8}}\Big(\sum_{g\in\{0,1\}^3}(-1)^{010\odot g}|g\rangle\Big) + \frac{1}{\sqrt{8}}\Big(\sum_{g\in\{0,1\}^3}(-1)^{011\odot g}|g\rangle\Big)\Big) \\
&= \frac{1}{\sqrt{2}}\Big(\frac{1}{\sqrt{8}}\Big(|000\rangle + |001\rangle - |010\rangle - |011\rangle + |100\rangle + |101\rangle - |110\rangle - |111\rangle\Big) \\
&\quad + \frac{1}{\sqrt{8}}\Big(|000\rangle - |001\rangle - |010\rangle + |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle\Big)\Big) \\
&= \frac{1}{\sqrt{2}}\frac{1}{\sqrt{8}}\Big(2\cdot|000\rangle - 2\cdot|010\rangle + 2\cdot|100\rangle - 2\cdot|110\rangle\Big) \\
&= \frac{1}{2}\Big(|000\rangle - |010\rangle + |100\rangle - |110\rangle\Big).
\end{aligned}
$$

We now measure the quantum register, and obtain any state from the superposition in $|\psi_3\rangle$ with equal probability. Assume we measure the bitstring 010. We need to repeat this procedure until we measured $n-1=2$ linearly independent bitstrings.

Hence, we run the same quantum circuit again (the calculation works analogously to above), but this time we measure the bitstring 000. This does not give us any new information, because the bitstrings 010 and 000 are not linearly independent (we can express 000 as $010 \oplus 010$).

We run the same quantum circuit again, this time we measure the bitstring 010. This is again not helpful, because we measured 010 already. We run the quantum circuit again, but now we measure 110 in the end. The bitstrings 010 and 110 are linearly independent, so we (on classical hardware) set up a system of equations

$$
\begin{aligned}
0\cdot s_2 + 1\cdot s_1 + 0\cdot s_0 &\equiv 0 \pmod 2 \\
1\cdot s_2 + 1\cdot s_1 + 0\cdot s_0 &\equiv 0 \pmod 2.
\end{aligned}
$$

The system has the trivial solution 000 and exactly one non-trivial solution $001 = s$. To verify that ρ is not bijective, we prepare two quantum registers in the state $|\psi_4\rangle = |000\rangle|00\rangle$ and run $\mathcal{U}_\rho|\psi_4\rangle = |000\rangle|00\rangle$. We measure the second register (which corresponds to $\rho(000)$) and measure the bitstring 00. Next, we prepare two quantum registers in the state $|\psi_5\rangle = |001\rangle|00\rangle$ and run $\mathcal{U}_\rho|\psi_5\rangle = |001\rangle|00\rangle$. We again measure the second register (which corresponds to $\rho(001)$) and measure the bitstring 00. Because $\rho(000) = \rho(001) = 00$, we conclude that ρ fulfills Simon's promise and its secret string s is 001.

4.4 Computational Complexity

As already explained, we use the quantum part of Simon's algorithm to generate bitstrings h such that $h\odot s = 0$ (where s is the secret string in Simon's problem). Since the algorithm

is probabilistic, we cannot give a number of executions after which we are guaranteed to have generated enough such elements. However, as is common for probabilistic algorithms, we can give an *expected* number of executions. In order to do so, we must establish how many different bitstrings $h \in \{0,1\}^n$ exist with $h \odot s = 0$, and we need to know with what probability any given set of such vectors is linearly independent.

Regarding the first point, we note that the set of all bitstrings of length n can be interpreted as a vector space over $\mathbb{Z}_2$ with dimension n and $\mathbf{0}$ as the neutral element. The $\oplus$ operation is the vector addition, and we define scalar-vector multiplication as $0 \cdot x = \mathbf{0}$ and $1 \cdot x = x$ for each $x \in \{0,1\}^n$. Furthermore, the elements x of that vector space that additionally satisfy $s \odot x = 0$ form a closed subspace, since by Lemma 1, for two elements $x, x' \in \{0,1\}^n$ with $x \odot s = x' \odot s = 0$ we have $s \odot (x \oplus x') = (s \odot x) \oplus (s \odot x') = 0$. In the following lemma, we analyze the dimension of the introduced subspace.

Lemma 4. *Let V be a subspace of $G = \{0,1\}^n$ such that for all $y \in V$ we have that $s \odot y = 0$. Then the dimension of that subspace is $n-1$.*

Proof. Consider the linear operator $A = \begin{pmatrix} s_{n-1} & s_{n-2} & \dots & s_0 \end{pmatrix}$. Since, for arbitrary $x \in G$ we have $Ax = \begin{pmatrix} s_{n-1}x_{n-1} \oplus s_{n-2}x_{n-2} \oplus \dots \oplus s_0 x_0 \end{pmatrix} = (s \odot x)$, we immediately see that $x \odot s \iff x \in \ker(A)$. We instantiate the Fundamental Theorem of Linear Maps and get $\dim(G) = \dim(\ker(A)) + \operatorname{rank}(A)$ and hence $n = \dim(\ker(A)) + 1$ and $\dim(\ker(A)) = n - 1$. □

Regarding the second point, in the following lemma, we calculate a lower bound for the probability that a set of elements drawn from a vector space over $\mathbb{Z}_2$ is linearly independent.

Lemma 5. *Let V be a vector space over $\mathbb{Z}_2$ with $\dim(V) = m$. Let $y^{(1)}, \dots, y^{(m)}$ be some elements of V sampled uniformly at random. Then the probability that $y^{(1)}, \dots, y^{(m)}$ are linearly independent is strictly greater than $\frac{1}{4}$.*

Proof. Let $\overline{P_{h^{(i)}}}$ denote the probability that the vectors $h^{(1)}, \dots, h^{(i)}$ are not linearly independent. We choose $y^{(1)}$ uniformly at random and analyze $\overline{P_{h^{(1)}}}$, the probability that $h^{(1)}$ is not linearly independent from itself. This is the case when $y^{(1)} = \mathbf{0}$, and the probability for this is $\frac{1}{2^m}$. We choose $y^{(2)}$ uniformly at random and analyze the probability $\overline{P_{h^{(2)}}}$, which is the probability for the event where $y^{(2)} = \mathbf{0}$ or $y^{(2)} = y^{(1)}$. Hence, we have that $\overline{P_{h^{(2)}}} = \frac{1}{2^m} + \frac{1}{2^m} = \frac{1}{2^{m-1}}$. We choose $y^{(3)}$ uniformly at random and analyze $\overline{P_{h^{(3)}}}$, the probability for the event where $h^{(3)} \in \{\mathbf{0}, h^{(1)}, h^{(2)}, h^{(1)} \oplus h^{(2)}\}$. We calculate $\overline{P_{h^{(3)}}} = \frac{1}{2^m} + \frac{1}{2^m} + \frac{1}{2^m} + \frac{1}{2^m} = \frac{1}{2^{m-2}}$.

Generalizing the above considerations, we calculate

$$\overline{P_{h^{(m-1)}}} = \overline{P_{h^{(1)}}} + \overline{P_{h^{(2)}}} + \dots + \overline{P_{h^{(m-3)}}} + \overline{P_{h^{(m-2)}}} = \frac{1}{2^m} + \frac{1}{2^{m-1}} + \dots + \frac{1}{8} + \frac{1}{4} = \sum_{k=2}^{m} \frac{1}{2^k}.$$

Using properties of the well-known geometric series we get

$$2 = \sum_{k=0}^{\infty} \frac{1}{2^k} = 1 + \frac{1}{2} + \sum_{k=2}^{\infty} \frac{1}{2^k}, \text{ and hence } \sum_{k=2}^{\infty} \frac{1}{2^k} = \frac{1}{2}.$$

The equality just derived helps us to more concretely estimate $\overline{P_{h^{(m-1)}}}$ to

$$\overline{P_{h^{(m-1)}}} = \sum_{k=2}^{m} \frac{1}{2^k} < \sum_{k=2}^{\infty} \frac{1}{2^k} = \frac{1}{2}.$$

As a consequence, the complementary probability $P_{h^{(m-1)}} = 1 - \overline{P_{h^{(m-1)}}}$, i.e. the probability that $h^{(1)}, \ldots, h^{(m-1)}$ are in fact linearly independent, is strictly larger than $\frac{1}{2}$. Assume now we already drew $m-1$ linearly independent vectors. Then the probability that the next vector we draw, $h^{(m)}$, is linearly independent from the first $m-1$ vectors is given by $\frac{2^{m-1}}{2^m} = \frac{1}{2}$. Hence, we know

$$P_{h^{(m)}} = P_{h^{(m-1)}} \cdot \frac{1}{2} > \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}.$$ □

Putting everything together, the elements $h \in \{0,1\}^n$ that satisfy $h \odot s = 0$ form a vector space of dimension $n-1$ (by Lemma 4), and hence by Lemma 5, when we draw $n-1$ elements from that vector space, they are linearly independent with probability strictly grater than $\frac{1}{4}$. Consequently, after an expected $4 \cdot (n-1) = O(n)$ runs of the quantum part of Simon's algorithm, we will have collected $n-1$ linearly independent bitstrings h with $h \odot s = 0$ with high probability.

By Theorem 2, the quantum circuit for Simon's algorithm, which we run $4 \cdot (n-1)$ times, has depth 3 plus the circuit depth of $\mathcal{U}_\rho$, hence the entire quantum part of Simon's algorithm has the expected circuit depth of $4 \cdot (n-1)$ times the circuit depth of $\mathcal{U}_\rho$ plus $4 \cdot (n-1) \cdot 3$.

We also need to classically post-process our linearly independent set of bitstrings. More precisely, we need to solve the system of equations induced by those bitstrings. A common algorithm for this task is *Gaussian elimination*, which, for $n-1$ bitstrings of length n can be done within $O(n^3)$ computation steps [19].

When we combine the circuit depth result for the quantum part and the amount of computation steps needed for the classical post post-processing, we end up with expected linearly many calls to the oracle implementation $\mathcal{U}_\rho$ and a number of other computation steps polynomial in n. In terms of complexity classes, we can thus place Simon's algorithm in $\mathcal{BQP}^{\mathcal{U}_\rho}$. This is faster than any known classical algorithm, be it determinstic or randomized [32], but not necessarily to an extent that warrants the expensive development of quantum hardware. However, when we compare the number of calls to the oracle for ρ only, Simon's hybrid quantum-classical routine is exponentially faster than any known purely classical algorithm [32].

CHAPTER 5

A Randomized Algorithm for the Extended Version of Simon's Problem

As mentioned in the introduction, Simon's problem was rephrased and generalized by Brassard and Høyer [13]. We will call this the extended version of Simon's problem, and it is formally defined as follows.

- **Input**: A black box implementation for some function $\rho : G \to R$ where
  - $G = \langle \{0,1\}^n, \oplus \rangle$, that is, the group over bit strings of length n, together with bitwise XOR (denoted by $\oplus$) as the group operation, and
  - R is some arbitrary set.

  Furthermore, ρ fulfills the promise that
  - there exists a subgroup $H \leq G$, and
  - $\forall g_1, g_2 \in G : \quad \rho(g_1) = \rho(g_2) \iff g_1 \oplus H = g_2 \oplus H$.
    That is, ρ behaves constant and distinct for each coset of H in G.
- **Output**: A generating set for H.

We first relate the standard version of Simon's problem from the previous chapter to this new version and show that the new version is a strict generalization. Consider an instance of Simon's problem where we get a black box implementation of some function ρ where ρ is bijective. We first reduce this instance of the original version to an instance of the extended version.

Lemma 6. *Let H be a trivial subgroup of $G = \langle \{0,1\}^n, \oplus \rangle$. Then the cosets of H in G are $\{g\}$ for each $g \in G$.*

Proof. Since H is a trivial subgroup of G, we can write $H = \{\mathbf{0}\}$. Then, by definition of a coset, for each $g \in G$, the coset of H induced by g can be written as

$$\{g \oplus h \mid h \in H\} = \{g \oplus \mathbf{0}\} = \{g\}.$$ □

If ρ is a bijective instance of the original version of Simon's problem, we can reduce it to an instance of the extended version of Simon's problem where the hidden subgroup H is trivial. The reason for this is that by Lemma 6, if ρ is bijective, then it behaves constant and distinct for each coset of the trivial subgroup $H = \{\mathbf{0}\}$ of G. The generating set is $\{\mathbf{0}\}$ in that case.

When, for an instance of Simon's problem, ρ is not bijective, this corresponds to an instance of the extended version of Simon's problem where the hidden subgroup H satisfies the property $|H| = 2$. Analogous to the first case, we examine the structure of the cosets of H in G.

Lemma 7. *Let H be any subgroup of $G = \langle \{0,1\}^n, \oplus \rangle$ with $|H| = 2$. Then H can be written as $\{\mathbf{0}, x\}$, where x is some non-trivial bitstring. Furthermore, for each $g \in G$, the coset of H in G induced by g has the form $\{g, g \oplus x\}$.*

Proof. For the first part of the lemma, we recall that because H is a subgroup of G, it must contain the neutral element from G, which is the bitstring $\mathbf{0}$. Because $|H| = 2$, H must contain one additional element x with $x \neq \mathbf{0}$, which is the first thing we needed to prove.

The second part of the lemma follows from the definition of a coset of H in G. For arbitrary $g \in G$, the coset induced by g has the form

$$\{g \oplus h \mid h \in H\} = \{g \oplus \mathbf{0}, g \oplus x\} = \{g, g \oplus x\}.$$ □

Consider an instance of the original version of Simon's problem where ρ is a function such that there exists a secret bitstring s and for all $g, g' \in G, g \neq g'$ we have that $\rho(g) = \rho(g') \iff g' = g \oplus s$. We reduce this to an instance of the extended version of Simon's problem by constructing the subgroup $H = \{\mathbf{0}, s\}$. By Lemma 7, ρ satisfying Simon's promise corresponds precisely to ρ behaving constant and distinct on each coset of H in G. It is easy to see that $\{s\}$ is a generating set for $H = \{\mathbf{0}, s\}$.

5.1 Quantum Algorithm for the Extended Version of Simon's Problem

We can reuse the quantum algorithm originally proposed by Simon [32] as part of the quantum algorithm for the extended version of Simon's problem. In that new setting

however, we need to adapt its mathematical analysis. The fact that the hidden subgroup has unknown size makes things substantially more complicated, hence before reasoning about the algorithm, we need to state auxiliary lemmata. The first one is purely technical.

Lemma 8 (Group Shift Lemma)**.** *Let G be a group and let H be any subgroup of $G = \langle \{0,1\}^n, \oplus \rangle$. Let $\alpha_{h'}$ denote some term depending on any particular $h' \in H$. Then we have*

$$\sum_{h \in H} \alpha_{h \oplus h'} = \sum_{h \in H} \alpha_h.$$

Proof. We note that by semantics of $\oplus$ and because H is a group and therefore closed we have $H = \{h \oplus h' \mid h \in H\}$. Hence, the difference between $\sum_{h \in H} \alpha_{h \oplus h'}$ and $\sum_{h \in H} \alpha_h$ is just the order of the summands. Since both sums are finite, the order does not influence the end result of the summation process. □

Algorithm 4.1, for simplification of the mathematical analysis, contains a measurement operation in step 4. This is the step where we measure and discard the second register holding the function output of ρ, so this step also abstracts away a lot of complexity regarding cosets (recall that the output of ρ is not random, but depends on the cosets of H in G). In the setting of the extended version of Simon's problem, we cannot perform that simplification anymore, hence now we need some more advanced mathematical machinery. The following lemma gives us additional information about cosets.

Lemma 9 (Coset Lemma)**.** *Let H be any subgroup of the group $G = \langle \{0,1\}^n, \oplus \rangle$. Then the following statements hold.*

- *The cosets of H in G partition G.*
- *There are precisely $\frac{|G|}{|H|}$ cosets of H in G.*

Proof. In order to prove that the cosets of H in G partition G, we first show that all elements of G are also elements of some coset of H in G. Second we show that two cosets are either equal or disjoint, i.e. no $g \in G$ is element of two distinct cosets.

By construction $\mathbf{0} \in H$ and hence every $g \in G$ is contained in the set $\{g \oplus h \mid h \in H\}$, which is the coset of H in G induced by g.

The fact that cosets are either equal or disjoint is a well known result in group theory, and we loosely follow the proof given by Scott [30]. Consider $t_1, t_2 \in G$ and assume that the two cosets $(t_1 \oplus H)$ and $(t_2 \oplus H)$ are not disjoint. Then there exists some bitstring c such that $c \in (t_1 \oplus H)$ and $c \in (t_2 \oplus H)$. Hence, we can write $c = t_1 \oplus h$ and $c = t_2 \oplus h'$ for some $h, h' \in H$. In the first equation, we XOR h on both sides, which gives us $t_1 = c \oplus h$. Now we replace c with $t_2 \oplus h'$ from the second equation and get $t_1 = t_2 \oplus h' \oplus h$. Because both h and $h' \in H$, by the group closure property, $h' \oplus h \in H$ and hence $t_1 \in (t_2 \oplus H)$.

Analogously we can infer $t_2 \in (t_1 \oplus H)$. The representatives t_1 and t_2 are chosen arbitrarily, so if any two cosets overlap, they are equal.

Last, we prove the claim about the coset count. For this we first analyze the size of any arbitrary coset $(g \oplus H)$. Suppose $|(g \oplus H)| < |H|$. Then there must exist at least two distinct bitstrings $h, h' \in H$ such that $g \oplus h = z$ and $g \oplus h' = z$ for some z. In that situation however we know $g \oplus h = g \oplus h'$ and hence $h = h'$, which is a contradiction to h and h' being distinct. Hence we must have $|g \oplus H| \geq |H|$. On the other hand, we must also have $|(g \oplus H)| \leq |H|$ since $((g \oplus H))$ is made up of the results of precisely $|H|$ applications of the $\oplus$ function, and we cannot obtain more than $|H|$ distinct results there. Hence, $|(g \oplus H)| = |H|$.

We also just proved that the cosets of H in G partition G, so each partition must have the same size and the total number of partitions must be $\frac{|G|}{|H|}$. □

Corollary 1. *If $t \oplus H$ is a coset of H in G, then for each $x \in (t \oplus H)$ it holds that $(x \oplus H) = (t \oplus H)$.*

Proof. The corollary follows directly from the fact that two cosets are either equal or disjoint and that for all x we have that $x \in (x \oplus H)$. □

Since in the original version of Simon's problem, the hidden subgroup H has a fixed size, the orthogonal set $H^\perp$ has a fixed size as well. In the extended version of Simon's problem, where H can be any subgroup of G, the size of $H^\perp$ also changes in response. This again forces us to make our analysis more sophisticated, and to address the added complexity. We state a more general version of Lemma 4.

Lemma 10 (Orthogonal Group Size Lemma)**.** *Let H be a subgroup of the group $G = \langle \{0,1\}^n, \oplus \rangle$ and let $H^\perp$ be the orthogonal set corresponding to H. Then we have $|H^\perp| = \frac{|G|}{|H|}$.*

Proof. Let $Y = \{y^{(1)}, \ldots, y^{(k)}\}$ be a basis of H, that is $\langle Y \rangle = H$. We use Y to construct a linear operator A in the following way

$$A = \begin{pmatrix} y_{n-1}^{(1)} & y_{n-2}^{(1)} & \cdots & y_0^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ y_{n-1}^{(k)} & y_{n-2}^{(k)} & \cdots & y_0^{(k)} \end{pmatrix}.$$

Note that because Y is a basis of H, A has the rank k.

We claim that for arbitrary $x \in G$, we have $x \in \ker(A) \iff x \in H^\perp$. The only-if direction is immediately obvious when we examine the result after we apply A to x (all

calculations are done in $\mathbb{Z}_2$, and x is interpreted as a column vector).

$$Ax = \begin{pmatrix} y_{n-1}^{(1)} x_{n-1} \oplus y_{n-2}^{(1)} x_{n-2} \oplus \cdots \oplus y_0^{(1)} x_0 \\ \cdots \\ y_{n-1}^{(k)} x_{n-1} \oplus y_{n-2}^{(k)} x_{n-2} \oplus \cdots \oplus y_0^{(k)} x_0 \end{pmatrix} = \begin{pmatrix} y^{(1)} \odot x \\ \cdots \\ y^{(k)} \odot x \end{pmatrix}.$$

If $x \in H^\perp$, then we have $x \odot h = 0$ for all $h \in H$ and in particular for all elements in Y.

For the if direction, assume towards a contradiction that $x \in \ker(A)$ but $x \notin H^\perp$. By the assumption $x \in \ker(A)$, we know that for all elements $y^{(i)}$ of Y, $x \odot y^{(i)} = 0$. By the assumption that $x \notin H^\perp$, there then exists some element $y \in H$ such that $x \odot y = 1$. If $y \in Y$, we have a contradiction to the assumption that $x \in \ker(A)$. If $y \notin Y$, then, because Y is a basis of H, we can write $y = y^{(i_1)} \oplus \cdots \oplus y^{(i_\ell)}$ where $y^{(i_1)}, \ldots, y^{(i_\ell)} \in Y$. This again leads to a contradiction since by the Distributivity Lemma (Lemma 1) we have

$$x \odot y = x \odot (y^{(i_1)} \oplus \cdots \oplus y^{(i_\ell)}) = \underbrace{(x \odot y^{(i_1)})}_{0 \text{ since } x \in \ker(A)} \oplus \cdots \oplus \underbrace{(x \odot y^{(i_\ell)})}_{0 \text{ since } x \in \ker(A)} = 0.$$

Now we instantiate the Fundamental Theorem of Linear Maps (Theorem 1). We have that $\dim(G) = \dim(\ker(A)) + \operatorname{rank}(A)$ and consequently $n = \dim(H^\perp) + k$ and thus $\dim(H^\perp) = n - k$. In our situation, where the vector space or the group we are working in are bitstrings, if $\dim(H^\perp) = n - k$, then $|H^\perp| = 2^{n-k}$. Furthermore, because $|Y| = k$ and $\langle Y \rangle = H$, we have $|H| = 2^k$. Thus, we write

$$|H^\perp| = 2^{n-k} = \frac{2^n}{2^k} = \frac{|G|}{|H|}.$$ □

Finally, we state Simon's quantum algorithm again in Algorithm 5.1, but this time precisely as it was stated by Simon himself [32]. That is, when compared to Algorithm 4.1, we skip step 4 where we measure the second register.

Algorithm 5.1: Simon's Algorithm

1. $|\psi_0\rangle = |0\rangle^{\otimes n} |0\rangle^{\otimes m}$ — initial state
2. $|\psi_1\rangle = (\mathcal{H}^{\otimes n} \otimes \mathcal{I}^{\otimes m}) |\psi_0\rangle$ — apply $\mathcal{H}$ to each qubit in first register
3. $|\psi_2\rangle = \mathcal{U}_\rho |\psi_1\rangle$ — apply oracle for ρ
4. $|\psi_3\rangle = (\mathcal{H}^{\otimes n} \otimes \mathcal{I}^{\otimes m}) |\psi_2\rangle$ — apply $\mathcal{H}$ to each qubit in first register
5. Obtain h by measuring the first register.

We are now equipped to analyze the behavior of Simon's algorithm in the setting of the extended version of Simon's problem. The following Theorem 3 and its proof are based

on Section 3 in the paper of Brassard and Høyer [13]. Note that while in the proof of Theorem 2 we reason about Algorithm 4.1, in Theorem 3 we reason about Algorithm 5.1.

Theorem 3. *Let $G = \langle \{0,1\}^n, \oplus \rangle$ and H be any subgroup of G. Furthermore, let $\rho : G \to \{0,1\}^m$ be a function that fulfills the promise of the extended version of Simon's problem: ρ behaves constant and distinct on each coset of H. Let $\mathcal{U}_\rho$ be a quantum implementation of ρ, that is $\mathcal{U}_\rho |g\rangle |g'\rangle = |g\rangle |g' \oplus \rho(g)\rangle$ for all $g \in G$ and $g' \in \{0,1\}^m$. Let T be a set composed of precisely one representative for each coset of H in G. Then Algorithm 5.1 performs the state change*

$$|0\rangle^{\otimes n} |0\rangle^{\otimes m} \to \frac{1}{\sqrt{|T|}} \Big(\sum_{t \in T} \Big| \phi_t H^\perp \Big\rangle |\rho(t)\rangle \Big).$$

Proof. Following the presentation of Brassard and Høyer [13], we start in the state

$$|\psi_0\rangle = |0\rangle^{\otimes n} |0\rangle^{\otimes m}.$$

We first apply the Hadamard transformation to the first register, resulting in the state

$$|\psi_1\rangle = \mathcal{H}^{\otimes n} \otimes \mathcal{I}^{\otimes m} \ \ |\psi_0\rangle = \frac{1}{\sqrt{|G|}} \Big(\sum_{g \in G} |g\rangle |0\rangle^{\otimes m} \Big).$$

Next, we apply $\mathcal{U}_\rho$ resulting in

$$|\psi_2\rangle = \mathcal{U}_\rho \ \ |\psi_1\rangle = \frac{1}{\sqrt{|G|}} \Big(\sum_{g \in G} |g\rangle |\mathbf{0} \oplus \rho(g)\rangle \Big) = \frac{1}{\sqrt{|G|}} \Big(\sum_{g \in G} |g\rangle |\rho(g)\rangle \Big).$$

Let $T^{(1)}, \ldots, T^{(|T|)}$ be the cosets of H in G. By Lemma 9, the cosets of H in G partition G and thus we can rewrite $|\psi_2\rangle$ into

$$|\psi_2\rangle = \frac{1}{\sqrt{|G|}} \Big(\sum_{t \in T^{(1)}} |t\rangle |\rho(t)\rangle + \cdots + \sum_{t \in T^{(|T|)}} |t\rangle |\rho(t)\rangle \Big).$$

Next we use Corollary 1 to pick one representative $t^{(j)}$ for each coset $T^{(j)}$ and write $T^{(j)}$ as $t^{(j)} \oplus H$. The result is

$$|\psi_2\rangle = \frac{1}{\sqrt{|G|}} \Big(\sum_{h \in H} \Big| t^{(1)} \oplus h \Big\rangle \Big| \rho(t^{(1)} \oplus h) \Big\rangle + \cdots + \sum_{h \in H} \Big| t^{(|H|)} \oplus h \Big\rangle \Big| \rho(t^{(|H|)} \oplus h) \Big\rangle \Big).$$

Because $T^{(j)} = t^{(j)} \oplus H$ and hence $(t^{(j)} \oplus h) \in T^{(j)}$ for each $h \in H$, we can use the promise of ρ to simplify the second register. Recall that ρ behaves constant and distinct for each coset of H in G, and hence $\rho(t^{(j)} \oplus h) = \rho(t^{(j)})$ for each $h \in H$. When we apply this to $|\psi_2\rangle$ we get

$$\begin{aligned} |\psi_2\rangle &= \frac{1}{\sqrt{|G|}} \Big(\sum_{h \in H} \Big| t^{(1)} \oplus h \Big\rangle \Big| \rho(t^{(1)}) \Big\rangle + \cdots + \sum_{h \in H} \Big| t^{(|H|)} \oplus h \Big\rangle \Big| \rho(t^{(|H|)}) \Big\rangle \Big) \\ &= \frac{1}{\sqrt{\frac{|G|}{|H|}}} \Big(\frac{1}{\sqrt{|H|}} \sum_{h \in H} \Big| t^{(1)} \oplus h \Big\rangle \Big| \rho(t^{(1)}) \Big\rangle + \cdots + \frac{1}{\sqrt{|H|}} \sum_{h \in H} \Big| t^{(|H|)} \oplus h \Big\rangle \Big| \rho(t^{(|H|)}) \Big\rangle \Big). \end{aligned}$$

Now recall the definition of the shorthand notation $\frac{1}{\sqrt{|H|}}\sum_{h\in H}|t\oplus h\rangle = |t\oplus H\rangle$. We use it to simplify $|\psi_2\rangle$ into

$$|\psi_2\rangle = \frac{1}{\sqrt{\frac{|G|}{|H|}}}\Big(\Big|t^{(1)}\oplus H\Big\rangle\Big|\rho(t^{(1)})\Big\rangle + \cdots + \Big|t^{(|T|)}\oplus H\Big\rangle\Big|\rho(t^{(|T|)})\Big\rangle\Big).$$

Using $|T| = \frac{|G|}{|H|}$ from Lemma 9, we rewrite $|\psi_2\rangle$ a final time into

$$|\psi_2\rangle = \frac{1}{\sqrt{|T|}}\Big(\sum_{t\in T}|t\oplus H\rangle\,|\rho(t)\rangle\Big),$$

which is the superposition derived by Brassard and Høyer in their description of Simon's algorithm in Section 3 in [13]. Our final quantum operation is to apply the Hadamard transformation to the first register again. Using the definition of the Hadamard transformation $\mathcal{H}^{\otimes n}|x\rangle = \frac{1}{\sqrt{|G|}}\sum_{g\in G}(-1)^{g\odot x}|g\rangle$ we get

$$\begin{aligned}
|\psi_3\rangle = \mathcal{H}^{\otimes n}\otimes\mathcal{I}^{\otimes m}\ \ |\psi_2\rangle &= \frac{1}{\sqrt{|T|}}\Big(\sum_{t\in T}\mathcal{H}^{\otimes n}|t\oplus H\rangle\,|\rho(t)\rangle\Big)\\
&= \frac{1}{\sqrt{|T|}}\Big(\sum_{t\in T}\frac{1}{\sqrt{|H|}}\Big(\sum_{h\in H}\mathcal{H}^{\otimes n}|t\oplus h\rangle\Big)\Big)|\rho(t)\rangle\\
&= \frac{1}{\sqrt{|T|}}\Big(\sum_{t\in T}\frac{1}{\sqrt{|H|}}\Big(\sum_{h\in H}\frac{1}{\sqrt{|G|}}\Big(\sum_{g\in G}(-1)^{g\odot(t\oplus h)}|g\rangle\Big)\Big)\Big)|\rho(t)\rangle\,.
\end{aligned}$$

We apply the Distributivity Lemma (Lemma 1) to rewrite ψ_3 into

$$|\psi_3\rangle = \frac{1}{\sqrt{|T|}}\Big(\sum_{t\in T}\frac{1}{\sqrt{|H|}}\Big(\sum_{h\in H}\frac{1}{\sqrt{|G|}}\Big(\sum_{g\in G}(-1)^{g\odot t}(-1)^{g\odot h}|g\rangle\Big)\Big)\Big)|\rho(t)\rangle\,.$$

We now perform a case distinction on H. Suppose H is non-trivial and recall the definition of $H^\perp = \{g\in G \mid g\odot h = 0\ \ \forall h\in H\}$. Following a proof idea by Mihara and Sung [26], we consider the superposition the first register is in and fix an arbitrary $g\notin H^\perp$. Such a g must exist, because $|H|>1$ and hence, by the Orthogonal Group Size Lemma (Lemma 10), $|H^\perp|<|G|$. The amplitude of $|g\rangle$ is given by

$$\begin{aligned}
&\frac{1}{\sqrt{|T|}}\Big(\sum_{t\in T}\frac{1}{\sqrt{|H|}}\Big(\sum_{h\in H}\frac{1}{\sqrt{|G|}}(-1)^{g\odot t}(-1)^{g\odot h}\Big)\Big)\\
=&\frac{1}{\sqrt{|T|}}\Big(\sum_{t\in T}\frac{1}{2\sqrt{|H|}}\Big(\sum_{h\in H}\frac{1}{\sqrt{|G|}}(-1)^{g\odot t}(-1)^{g\odot h} + \sum_{h\in H}\frac{1}{\sqrt{|G|}}(-1)^{g\odot t}(-1)^{g\odot h}\Big)\Big).
\end{aligned}$$

By construction of $H^\perp$ and because $g \notin H^\perp$, there exists some $h' \in H$ with $g \odot h' = 1$. We apply the Group Shift Lemma (Lemma 8) to rewrite the amplitude of g further into

$$\frac{1}{\sqrt{|T|}}\left(\sum_{t\in T}\frac{1}{2\sqrt{|H|}}\left(\sum_{h\in H}\frac{1}{\sqrt{|G|}}(-1)^{g\odot t}(-1)^{g\odot h}+\sum_{h\in H}\frac{1}{\sqrt{|G|}}(-1)^{g\odot t}(-1)^{g\odot(h\oplus h')}\right)\right)$$
$$=\frac{1}{\sqrt{|T|}}\left(\sum_{t\in T}\frac{1}{2\sqrt{|H|}}\frac{1}{\sqrt{|G|}}\left(\sum_{h\in H}(-1)^{g\odot t}(-1)^{g\odot h}+(-1)^{g\odot t}(-1)^{g\odot(h\oplus h')}\right)\right).$$

We use Lemma 1 again to rewrite the amplitude to

$$\frac{1}{\sqrt{|T|}}\left(\sum_{t\in T}\frac{1}{2\sqrt{|H|}}\frac{1}{\sqrt{|G|}}\left(\sum_{h\in H}(-1)^{g\odot t}(-1)^{g\odot h}+(-1)^{g\odot t}(-1)^{g\odot h}(-1)^{g\odot h'}\right)\right)$$
$$=\frac{1}{\sqrt{|T|}}\left(\sum_{t\in T}\frac{1}{2\sqrt{|H|}}\frac{1}{\sqrt{|G|}}\left(\sum_{h\in H}(-1)^{g\odot t}(-1)^{g\odot h}\left(1+(-1)^{g\odot h'}\right)\right)\right).$$

Now we recall that $g \odot h' = 1$ to conclude that the amplitude of g is

$$\frac{1}{\sqrt{|T|}}\left(\sum_{t\in T}\frac{1}{2\sqrt{|H|}}\frac{1}{\sqrt{|G|}}\left(\sum_{h\in H}(-1)^{g\odot t}(-1)^{g\odot h}\left(1-1\right)\right)\right)=0.$$

The innermost sum in $|\psi_3\rangle$ is over elements of the group G. We just showed that for arbitrary $g \in G$ with $g \notin H^\perp$, the amplitude of g is zero. Hence, we can rewrite the innermost sum in $|\psi_3\rangle$ into a sum over elements of $H^\perp$ only. The result is

$$|\psi_3\rangle=\frac{1}{\sqrt{|T|}}\left(\sum_{t\in T}\frac{1}{\sqrt{|H|}}\left(\sum_{h\in H}\frac{1}{\sqrt{|G|}}\left(\sum_{g\in H^\perp}(-1)^{g\odot t}(-1)^{g\odot h}\,|g\rangle\right)\right)\right)|\rho(t)\rangle\,.$$

We can perform the very same rewrite of $|\psi_3\rangle$ as well if H is trivial. In that case, by Lemma 10, we have $|H^\perp| = |G|$ and hence we can use G and $H^\perp$ interchangeably.

Recall that by definition of $H^\perp$ and by construction of h, for each element g in $H^\perp$ we have $g \odot h = 0$. We use this to simplify $|\psi_3\rangle$ into

$$|\psi_3\rangle=\frac{1}{\sqrt{|T|}}\left(\sum_{t\in T}\frac{1}{\sqrt{|H|}}\left(\sum_{h\in H}\frac{1}{\sqrt{|G|}}\left(\sum_{g\in H^\perp}(-1)^{g\odot t}\,|g\rangle\right)\right)\right)|\rho(t)\rangle\,.$$

Now we realize that h does not occur anymore in any summand and further rewrite $|\psi_3\rangle$ into

$$|\psi_3\rangle=\frac{1}{\sqrt{|T|}}\left(\sum_{t\in T}\frac{|H|}{\sqrt{|H|}}\frac{1}{\sqrt{|G|}}\left(\sum_{g\in H^\perp}(-1)^{g\odot t}\,|g\rangle\right)\right)|\rho(t)\rangle$$
$$=\frac{1}{\sqrt{|T|}}\left(\sum_{t\in T}\sqrt{\frac{|H|}{|G|}}\left(\sum_{g\in H^\perp}(-1)^{g\odot t}\,|g\rangle\right)\right)|\rho(t)\rangle\,.$$

Last, we apply Lemma 10 again to get $|H^\perp| = \frac{|G|}{|H|}$. Taking both sides of the equality to the power of -1 gives us $\frac{1}{|H^\perp|} = \frac{|H|}{|G|}$, which we use to rewrite $|\psi_3\rangle$ into

$$|\psi_3\rangle = \frac{1}{\sqrt{|T|}}\Big(\sum_{t\in T}\frac{1}{\sqrt{|H^\perp|}}\Big(\sum_{g\in H^\perp}(-1)^{g\odot t}\,|g\rangle\Big)\Big)\,|\rho(t)\rangle = \frac{1}{\sqrt{|T|}}\Big(\sum_{t\in T}\left|\phi_t H^\perp\right\rangle\,|\rho(t)\rangle\Big).$$

This is the superposition also stated in Equation (4) in the paper by Brassard and Høyer [13]. □

Note that the final superposition $|\psi_3\rangle$ is precisely what we get in the original version of Simon's problem (as presented in Chapter 4), just written down differently. Let ρ be an instance of the original version of Simon's problem such that ρ is bijective (i.e. the hidden subgroup is trivial). In that case, by Lemma 6, both sets T and $H^\perp$ would correspond to G, and we would measure some random bitstring. If ρ is not bijective (i.e. the hidden subgroup is of form $\{\mathbf{0}, s\}$), then $|\psi_3\rangle$ would be a superposition over all $g \in G$ with $g \odot s = 0$, which is exactly the definition of $H^\perp$.

In the original version of Simon's problem, we know the sizes of the possible hidden subgroups. Hence, we also know that we can stop sampling elements from $H^\perp$ when we have collected $n-1$ linearly independent ones, and we can construct a generating set for H. In the extended setting however, we do not know $|H|$ and hence we also do not know when to stop sampling. Moreover, we would like to not measure any already known elements from $H^\perp$ again. In the subsequent sections, we will discuss a way to achieve both.

CHAPTER 6

Removing Already Known Vectors from a Superposition

When we apply the original version of Simon's algorithm, it might happen that we measure the same element from $H^\perp$ multiple times. In their paper [13], Brassard and Høyer propose a quantum routine that prevents exactly that. By applying additional quantum operators (i.e. introducing *blocking clauses*), we can make sure to always measure a new element from $H^\perp$ if such an element exists.

We first present the Equal Cardinality Lemma, which is an important tool to reason about the structure of arbitrary subgroups of G. Next, we discuss a quantum algorithm that allows us to introduce a single blocking clause (that is, a quantum algorithm with which we ensure not to measure one particular known element from $H^\perp$ again) in the Blocking Clause Lemma. Finally, generalizing the Blocking Clause Lemma, we introduce a quantum algorithm that ensures that we do not measure a set of already known elements from $H^\perp$ again.

6.1 Implementing a Single Blocking Clause

The following lemma gives us a crucial insight into non-trivial subgroups of $\langle\{0,1\}^n, \oplus\rangle$, which we are going to exploint for the quantum algorithm that blocks already measured bitstrings from future measurements. Intuitively, we can split a non-trivial subgroup of G into two parts using some non-trivial pivot element and based on the structure of the pivot element, we get information on the structure of the two parts of the subgroup. A similar observation can be found in Section 4.3 in the paper by Brassard and Høyer [13], but there it is stated without a proof.

Lemma 11 (Equal Cardinality Lemma)**.** *Let H be a non-trivial subgroup of group $G = \langle\{0,1\}^n, \oplus\rangle$. Then for some j with $0 \leq j < n$, there exists an element $y \in H$ such*

that the two sets

$$K_j = \{k \in H \mid k_j = 0\} \quad \text{and} \quad K'_j = \{k' \in H \mid k'_j = 1\}$$

have the same cardinality, i.e. $|K_j| = |K'_j|$.

Proof. First note that there exists at least one non-zero element y in H, because H is non-trivial. Because y is non-zero, there exists a bit y_j in y with $y_j = 1$. Thus, the sets K_j and K'_j are well-defined.

Consider the following set $A = \{y \oplus k' \mid k' \in K'_j\}$. We note that for an arbitrary $k' \in K'_j$ we have $y_j = k'_j = 1$ and hence $(y \oplus k')_j = 0$ and thus $(y \oplus k') \in K_j$. Consequently, $A \subseteq K_j$. We note that for arbitrary $a, b \in G$ we have $(y \oplus a = y \oplus b) \implies a = b$ and thus $|A| = |K'_j|$. Consequently, we have

$$|K'_j| \leq |K_j|.$$

Analogously, consider the set $B = \{y \oplus k \mid k \in K_j\}$. Note that for an arbitrary $k \in K_j$, because $y_j = 1$ and $k_j = 0$, we have that $(y \oplus k)_j = 1$ and hence $(y \oplus k) \in K'_j$. Consequently, $B \subseteq K'_j$. Analogous to above, $|B| = |K_j|$ and we get

$$|K_j| \leq |K'_j|.$$

Then $|K_j| = |K'_j|$. □

Corollary 2. $K'_j = \{y \oplus k \mid k \in K_j\}$.

Proof. This follows immediately from the proof of the Equal Cardinality Lemma. Consider set B and recall $B \subseteq K'_j$ and $|B| = |K_j|$. Using $|K_j| = |K'_j|$ from the Equal Cardinality Lemma, we get $|B| = |K'_j|$ and therefore $K'_j = B = \{y \oplus k \mid k \in K_j\}$. □

Moving on, we present a quantum algorithm that implements a single blocking clause. Intuitively, we are given a quantum register in a superposition state that also includes an element from a subgroup of G which we already know and which we would like to not measure again. In the proof of the Blocking Clause Lemma below, we construct a quantum algorithm that manipulates that superposition in a way that our known element is not included anymore. Note that the Blocking Clause Lemma was originally stated by Brassard and Høyer as Lemma 6 [13]. Our presentation closely follows the original one, but we present the mathematical analysis in much more detail.

Lemma 12 (Blocking Clause Lemma)**.** *Let H be a non-trivial subgroup of group $G = \langle\{0,1\}^n, \oplus\rangle$. Furthermore, let y be a non-zero element of H. Define (as in Lemma 11) the two sets*

$$K_j = \{k \in H \mid k_j = 0\} \quad \text{and} \quad K'_j = \{k' \in H \mid k'_j = 1\}$$

with $y_j = 1$ for some j with $0 \leq j < n$. Then there exists a quantum algorithm that, applied to two quantum registers of size n and 1, performs the state change

$$|\phi_g H\rangle\,|0\rangle \to |\phi_g K_j\rangle\,|g \odot y\rangle\,.$$

Moreover, the algorithm can be implemented as a measurement-free quantum circuit of depth linear in n.

Proof. Assume two quantum registers, the first one of size n and the second one of size 1. Since H is non-trivial, there is at least one $y \in H$, with $y \neq \mathbf{0}$. Choose an arbitrary j for which $y_j = 1$ holds. Then we start in the state

$$|\psi_0\rangle = |\phi_g H\rangle\,|0\rangle = \frac{1}{\sqrt{|H|}} \sum_{h \in H} (-1)^{g \odot h}\,|h\rangle\,|0\rangle\,.$$

Following Brassard and Høyer [13], we apply the $\mathcal{CNOT}$operator with the j-th qubit in the first register as control qubit and the single qubit in the second register as target qubit. The resulting state is

$$|\psi_1\rangle = \frac{1}{\sqrt{|H|}} \Big(\sum_{h \in H} (-1)^{g \odot h}\,|h\rangle\,|h_j\rangle \Big).$$

What follows next is a complex rewriting of $|\psi_1\rangle$, which is not further detailed in [13]. We note that $K_j \cup K'_j = H$ and that $K_j \cap K'_j = \emptyset$. Thus, we can rewrite $|\phi_g H\rangle$ to $\big|\phi_g(K_j \cup K'_j)\big\rangle$. Then

$$\begin{aligned} |\psi_1\rangle &= \frac{1}{\sqrt{|(K_j \cup K'_j)|}} \Big(\sum_{h \in (K_j \cup K'_j)} (-1)^{g \odot h}\,|h\rangle\,|h_j\rangle \Big) \\ &= \frac{1}{\sqrt{|K_j \cup K'_j|}} \Big(\Big(\sum_{k \in K_j} (-1)^{g \odot k}\,|k\rangle\,|0\rangle \Big) + \Big(\sum_{k' \in K'_j} (-1)^{g \odot k'}\,|k'\rangle\,|1\rangle \Big) \Big). \end{aligned}$$

Exploiting the Equal Cardinality Lemma (Lemma 11), we derive the following identity

$$\frac{1}{\sqrt{|K_j \cup K'_j|}} = \frac{1}{\sqrt{|K_j| + |K'_j|}} = \frac{1}{\sqrt{|K_j| + |K_j|}} = \frac{1}{\sqrt{2 \cdot |K_j|}} = \frac{1}{\sqrt{2}} \cdot \frac{1}{\sqrt{|K_j|}}.$$

We apply that identity to $|\psi_1\rangle$ and get

$$|\psi_1\rangle = \frac{1}{\sqrt{2}} \frac{1}{\sqrt{|K_j|}} \Big(\Big(\sum_{k \in K_j} (-1)^{g \odot k}\,|k\rangle\,|0\rangle \Big) + \Big(\sum_{k' \in K'_j} (-1)^{g \odot k'}\,|k'\rangle\,|1\rangle \Big) \Big).$$

With $y_j = 1$ and Corollary 2, we rewrite the sum over K' into a sum over K. We get

$$|\psi_1\rangle = \frac{1}{\sqrt{2}} \frac{1}{\sqrt{|K_j|}} \Big(\Big(\sum_{k \in K_j} (-1)^{g \odot k}\,|k\rangle\,|0\rangle \Big) + \Big(\sum_{k \in K_j} (-1)^{g \odot (k \oplus y)}\,|y \oplus k\rangle\,|1\rangle \Big) \Big).$$

Next we apply the Distributivity Lemma (Lemma 1) and get

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}\frac{1}{\sqrt{|K_j|}}\left(\left(\sum_{k\in K_j}(-1)^{g\odot k}|k\rangle|0\rangle\right)+\left(\sum_{k\in K_j}(-1)^{g\odot k}(-1)^{g\odot y}|y\oplus k\rangle|1\rangle\right)\right)$$
$$= \frac{1}{\sqrt{2}}\frac{1}{\sqrt{|K_j|}}\left(\left(\sum_{k\in K_j}(-1)^{g\odot k}|k\rangle|0\rangle\right)+(-1)^{g\odot y}\left(\sum_{k\in K_j}(-1)^{g\odot k}|y\oplus k\rangle|1\rangle\right)\right).$$

For the next steps, let $\cdot$ denote scalar-vector multiplication. Recall that y denotes a bitstring, which we interpret as a vector. Hence, we have $1\cdot y = y$ and $0\cdot y = \mathbf{0}$. We note about $\mathbf{0}$ that $x\odot\mathbf{0} = 0$ and consequently $(-1)^{x\odot\mathbf{0}} = (-1)^0 = 1$ for arbitrary $x\in G$. We rewrite ψ_1 even further to

$$\begin{aligned}|\psi_1\rangle &= \frac{1}{\sqrt{2}}\frac{1}{\sqrt{|K_j|}}\Bigg((-1)^{g\odot(0\cdot y)}\left(\sum_{k\in K_j}(-1)^{g\odot k}|k\rangle|0\rangle\right)+\\ &\qquad (-1)^{g\odot(1\cdot y)}\left(\sum_{k\in K_j}(-1)^{g\odot k}|y\oplus k\rangle|1\rangle\right)\Bigg)\\ &= \frac{1}{\sqrt{2}}\frac{1}{\sqrt{|K_j|}}\Bigg((-1)^{g\odot(0\cdot y)}\left(\sum_{k\in K_j}(-1)^{g\odot k}|(0\cdot y)\oplus k\rangle|0\rangle\right)+\\ &\qquad (-1)^{g\odot(1\cdot y)}\left(\sum_{k\in K_j}(-1)^{g\odot k}|(1\cdot y)\oplus k\rangle|1\rangle\right)\Bigg)\\ &= \frac{1}{\sqrt{2}}\sum_{i\in\mathbb{Z}_2}(-1)^{g\odot(i\cdot y)}\left(\frac{1}{\sqrt{|K_j|}}\sum_{k\in K_j}(-1)^{g\odot k}|(i\cdot y)\oplus k\rangle\right)|i\rangle\,.\end{aligned}$$

This is the superposition stated in [13] on page 5. Following the presentation of Brassard and Høyer, we apply a quantum operation that performs the following state change on the first register if the second register is in state $|1\rangle$: $|x\rangle|1\rangle\to|x\oplus y\rangle|1\rangle$. We observe that if the second register is in state $|1\rangle$, the first register is in a superposition state over all elements $k\in K_j$ where each summand is of form $|(1\cdot y)\oplus k\rangle$. After the application of the quantum operation, by the semantics of $\oplus$, the first register of each summand is in the state $|k\rangle$, and the resulting overall state of our quantum registers is

$$\begin{aligned}|\psi_2\rangle &= \frac{1}{\sqrt{2}}\sum_{i\in\mathbb{Z}_2}(-1)^{g\odot(i\cdot y)}\left(\frac{1}{\sqrt{|K_j|}}\sum_{k\in K_j}(-1)^{g\odot k}|k\rangle\right)|i\rangle\\ &= \left(\frac{1}{\sqrt{|K_j|}}\sum_{k\in K_j}(-1)^{g\odot k}|k\rangle\right)\left(\frac{1}{\sqrt{2}}\sum_{i\in\mathbb{Z}_2}(-1)^{g\odot(i\cdot y)}|i\rangle\right)\\ &= |\phi_g K_j\rangle\left(\frac{1}{\sqrt{2}}\sum_{i\in\mathbb{Z}_2}(-1)^{g\odot(i\cdot y)}|i\rangle\right).\end{aligned}$$

From now on, we omit the first register, because its content does not change. Instead, we focus on the second register, which we can rewrite to

$$\begin{aligned}|\psi_2'\rangle &= \frac{1}{\sqrt{2}} \sum_{i \in \mathbb{Z}_2} (-1)^{g \odot (i \cdot y)} |i\rangle = \frac{1}{\sqrt{2}} \Big((-1)^{g \odot (0 \cdot y)} |0\rangle + (-1)^{g \odot (1 \cdot y)} |1\rangle \Big) \\ &= \frac{1}{\sqrt{2}} \Big((-1)^{(g \odot y) \cdot 0} |0\rangle + (-1)^{(g \odot y) \cdot 1} |1\rangle \Big) = \frac{1}{\sqrt{2}} \sum_{i \in \mathbb{Z}_2} (-1)^{(g \odot y) \cdot i} |i\rangle \\ &= \frac{1}{\sqrt{2}} \sum_{i \in \mathbb{Z}_2} (-1)^{(g \odot y) \odot i} |i\rangle \,.\end{aligned}$$

We instantiate the Hadamard identity $\mathcal{H}^{\otimes n} |x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{y \odot x} |y\rangle$ for $n = 1$ and get $\mathcal{H} |x\rangle = \frac{1}{\sqrt{2}} \sum_{y \in \{0,1\}^1} (-1)^{y \odot x} |y\rangle = \frac{1}{\sqrt{2}} \sum_{y \in \{0,1\}} (-1)^{y \cdot x} |y\rangle$, which we use to further simplify $|\psi_2'\rangle$ into

$$|\psi_2'\rangle = \mathcal{H} |g \odot y\rangle \,.$$

Following the presentation of Brassard and Høyer [13], we now apply the $\mathcal{H}$ operator to the second register. The resulting final state is

$$|\psi_3\rangle = \mathcal{H} |\psi_2\rangle = \mathcal{H}\mathcal{H} |g \odot y\rangle = \mathcal{I} |g \odot y\rangle = |g \odot y\rangle \,.$$

The overall state of our two quantum registers is thus $|\phi_g K_j\rangle |g \odot y\rangle$, which concludes the first part of the proof.

We still need to prove that there is a circuit with depth linear in n that performs the above algorithm. The state transition from $|\psi_0\rangle$ to $|\psi_1\rangle$ is performed using one $\mathcal{CNOT}$ operator only, so this part even has constant circuit depth. The transition from $|\psi_1\rangle$ to $|\psi_2\rangle$ can be implemented in the following way. We traverse each bit y_i in y and extend our circuit in case $y_1 = 1$. We add a *CNOT* gate where the control qubit is the single qubit in the second register and the target qubit is the i-th qubit in the first register.

Let our quantum registers be in state $|x\rangle |q\rangle$ before we apply the above circuit. Then we claim that this circuit changes the state of the registers to $|x \oplus y\rangle |q\rangle$ if and only if $|q\rangle = |1\rangle$. Suppose $|q\rangle = |0\rangle$. Then by construction, none of the *CNOT* gates in the above circuit get activated and thus the state of the register is not changed at all. Suppose $|q\rangle = |1\rangle$. By construction, each qubit $|x_i\rangle$ in the first register is in state $|x_i \oplus y_i\rangle$ after we apply the circuit. If $y_i = 0$, then we did not add any gate to the circuit and hence $|x_i\rangle = |x_i \oplus 0\rangle = |x_i \oplus y_i\rangle$. If $y_i = 1$, then the above circuit has a *CNOT* gate where the target qubit is $|x_i\rangle$ and the control qubit is $|q\rangle$. That *CNOT* gate gets activated as by assumption $|q\rangle = |1\rangle$. Hence, $|x_i\rangle$ gets changed to $|x_i \oplus 1\rangle = |x_i \oplus y_i\rangle$. Because each qubit $|x_i\rangle$ gets transformed to $|x_i \oplus y_i\rangle$, the entire value $|x\rangle$ gets changed to $|x \oplus y\rangle$. Note that we add a maximum of n *CNOT* gates for this step.

An application of a single Hadamard gate to state $|\psi_2\rangle$ results in state $|\psi_3\rangle$. The entire circuit is therefore built from $1 + O(n) + 1$ gates and does not apply any measurement operations. This concludes the second part of the proof. □

Corollary 3. *We can undo the effects of the circuit described in the Blocking Clause Lemma (Lemma 12) by just running it in reverse order.*

Proof. The corollary follows directly from the fact that all gates used in the above circuit implement self-adjoint quantum operators. □

Corollary 4. *We can generalize the setting for the above defined algorithm to the case where the second quantum register contains more than one qubit. That is, for all $\ell, 1 \leq \ell \leq m$, there exists a quantum algorithm $\mathcal{C}_\ell$ with*

$$\begin{aligned}&\mathcal{C}_\ell \left|\phi_g H\right\rangle \left|q_{m-1} \dots q_{m-\ell+1}\, 0\, q_{m-\ell-1} \dots q_0\right\rangle \\ &= \left|\phi_g K_j\right\rangle \left|q_{m-1} \dots q_{m-\ell+1} (g \odot y) q_{m-\ell-1} \dots q_0\right\rangle,\end{aligned}$$

where K_j is defined like in the Blocking Clause Lemma (Lemma 12). Moreover, $\mathcal{C}_\ell$ can be implemented as a quantum circuit of depth linear in n.

Proof. The proof works analogously to the proof of Lemma 12. For any $\mathcal{C}_\ell$, we just set the second quantum register from the Lemma 12 to be the ℓ-th most significant qubit from the second register in our setting. □

6.2 Implementing Multiple Blocking Clauses

We can repeatedly apply the algorithm from the Blocking Clause Lemma (Lemma 12) to exclude multiple already known states from a superposition. One precondition of the Blocking Clause Lemma is that our quantum register is in a superposition state over a group. We thus first prove in Lemma 13 that the superposition we get after we applied the Blocking Clause Lemma (Lemma 12), is still a superposition over a group.

Lemma 13. *Let H be any subgroup of the group $G = \{0,1\}^n$ and let $\{i_1, \dots, i_m\}$ be a set of indices where each element i_j satisfies $0 \leq j < n$. Define $K = \{h \in H \mid h_{i_1} = \dots = h_{i_m} = 0\}$. Then, $\langle K, \oplus \rangle$ is a group.*

Proof. We first note that K is non-empty because $\mathbf{0}$ is the neutral element of H, and we have $\mathbf{0}_{i_1} = \dots = \mathbf{0}_{i_m} = 0$. Hence, $\mathbf{0} \in K$. Next we prove the group axioms.

1. **Closure**: Fix arbitrary elements $k, k' \in K$. By definition of K we have $k_{i_1} = \dots = k_{i_m} = k'_{i_1} = \dots = k'_{i_m} = 0$. Hence, $(k \oplus k')_{i_1} = \dots = (k \oplus k')_{i_m} = 0$ and therefore $(k \oplus k') \in K$.
2. **Neutral Element**: The neutral element of K is $\mathbf{0}$ since for arbitrary $k \in K$ we have $k \oplus \mathbf{0} = k$.
3. **Inverse Element**: By definition of $\oplus$, each $k \in K$ is its own inverse.

4. **Associativity**: Follows directly from the definition of $\oplus$.

□

Second, we use Lemma 13 in a larger constructive proof for the existence of a quantum algorithm that shrinks a superposition to not include multiple already known elements. We formulate the Blocking Clause Theorem, which is adapted from Lemma 7 by Brassard and Høyer [13], who state it without a proof.

Theorem 4 (Blocking Clause Theorem)**.** *Let H be any subgroup of the group $G = \langle \{0,1\}^n, \oplus \rangle$, let $g \in G$, and let $Y = \{y^{(1)}, \ldots, y^{(|Y|)}\}$ be any subset of H where each $y^{(j)} \in Y$ satisfies the following two formulas*

- $\exists i_j : y^{(j)}_{i_j} = 1$, *and*
- $\forall \ell : 1 \leq \ell < j,\ y^{(j)}_{i_\ell} = 0$.

If $Y = \emptyset$ define $K_{(0)} = H$. Otherwise, for each j with $1 \leq j \leq |Y|$ define $K_{(j)} = \{h \in H \mid h_{i_1} = \cdots = h_{i_j} = 0\}$. Then there exists a quantum algorithm that performs the state change

$$|\phi_g H\rangle \, |0\rangle^{\otimes |Y|} \to \left|\phi_g K_{(|Y|)}\right\rangle \left(\bigotimes_{y^{(j)} \in Y} \left| g \odot y^{(j)} \right\rangle \right).$$

Moreover, the algorithm can be implemented as a quantum circuit with depth quadratic in n.

Proof. We first construct a quantum algorithm to perform the above state change. Second, we prove the correctness of that algorithm and third we prove the claim about its circuit depth.

Let $\mathcal{C}_0 = \mathcal{I}$ and for each j with $1 \leq j \leq |Y|$, let $\mathcal{C}_j$ be the quantum algorithm from Corollary 4 (where $y^{(j)}$ is the singled-out non-zero bitstring) and define $\mathcal{C}_Y = \mathcal{C}_{|Y|} \cdots \mathcal{C}_1 \mathcal{C}_0$. We prove the following statement via mathematical induction on $|Y|$.

$$\mathcal{C}_Y \, |\phi_g H\rangle \, |0\rangle^{\otimes |Y|} = \left|\phi_g K_{(|Y|)}\right\rangle \left(\bigotimes_{y^{(j)} \in Y} \left| g \odot y^{(j)} \right\rangle \right).$$

Induction Start $|Y| = 0$. Then $Y = \emptyset$ and by construction $\mathcal{C}_\emptyset = \mathcal{C}_0 = \mathcal{I}$. We have

$$\mathcal{C}_0 \, |\phi_g H\rangle \, |0\rangle^{\otimes 0} = \mathcal{I} \, |\phi_g H\rangle \, |0\rangle^{\otimes 0} = |\phi_g H\rangle \otimes (1) = \left|\phi_g K_{(0)}\right\rangle \left(\bigotimes_{y^{(j)} \in Y} \left| g \odot y^{(j)} \right\rangle \right),$$

where we used $|\psi\rangle^{\otimes 0} = (1)$ by definition of the tensor product.

Induction Hypothesis $|Y| = m - 1$ for some $m > 1$. Let $\mathcal{C}_Y = \mathcal{C}_{m-1} \cdots \mathcal{C}_0$ and

$$\mathcal{C}_Y \left|\phi_g H\right\rangle \left|0\right\rangle^{\otimes m-1} = \left(\mathcal{C}_{m-1} \cdots \mathcal{C}_0\right) \left|\phi_g H\right\rangle \left|0\right\rangle^{\otimes m-1} = \left|\phi_g K_{(m-1)}\right\rangle \left(\bigotimes_{y^{(j)} \in Y} \left|g \odot y^{(j)}\right\rangle \right).$$

Induction Step Consider Y with $|Y| = m$. Then $\mathcal{C}_Y = \mathcal{C}_m \cdots \mathcal{C}_1 \mathcal{C}_0$. Because $|Y| = m$, we can write $Y = \{y^{(1)}, \ldots, y^{(m)}\}$. When we apply $\mathcal{C}_Y$ to the initial state we get

$$\begin{aligned} \mathcal{C}_Y \left|\phi_g H\right\rangle \left|0\right\rangle^{\otimes m} &= \left(\mathcal{C}_m \mathcal{C}_{m-1} \cdots \mathcal{C}_0\right) \left|\phi_g H\right\rangle \left|0\right\rangle^{\otimes m} \\ &= \mathcal{C}_m \cdot \left(\mathcal{C}_{m-1} \cdots \mathcal{C}_0\right) \left|\phi_g H\right\rangle \left|0\right\rangle^{\otimes m-1} \left|0\right\rangle. \end{aligned}$$

We use the induction hypothesis to rewrite the above expression to

$$\mathcal{C}_m \left|\phi_g K_{(m-1)}\right\rangle \left(\bigotimes_{y^{(j)} \in Y \setminus \{y^{(m)}\}} \left|g \odot y^{(j)}\right\rangle \right) \left|0\right\rangle.$$

By construction of Y, there must exist some index i_m such that $y_{i_m}^{(m)} = 1$ and $y_{i_1}^{(m)} = \cdots = y_{i_{m-1}}^{(m)} = 0$. By Lemma 13, $K_{(m-1)}$ is a group and because $y^{(m)} \in K_{(m-1)}$, $K_{(m-1)}$ is non-trivial. Hence, we can apply the Blocking Clause Lemma (Lemma 12) to the above expression and obtain

$$\begin{aligned} &\mathcal{C}_m \left|\phi_g K_{(m-1)}\right\rangle \left(\bigotimes_{y^{(j)} \in Y \setminus \{y^{(m)}\}} \left|g \odot y^{(j)}\right\rangle \right) \left|0\right\rangle \\ &= \left|\phi_g K_{(m)}\right\rangle \left(\bigotimes_{y^{(j)} \in Y \setminus \{y^{(m)}\}} \left|g \odot y^{(j)}\right\rangle \right) \left|g \odot y^{(m)}\right\rangle \\ &= \left|\phi_g K_{(m)}\right\rangle \left(\bigotimes_{y^{(j)} \in Y} \left|g \odot y^{(j)}\right\rangle \right), \end{aligned}$$

which concludes the induction proof.

Having proven the correctness of $\mathcal{C}_Y$, we proceed by analyzing its circuit depth. By the Blocking Clause Lemma, for each j with $1 \leq j \leq |Y|$, $\mathcal{C}_j$ can be implemented as a quantum circuit of depth linear in n. We apply $|Y|$ such operators, so in order to reason about the total circuit depth, we need to give an upper bound on $|Y|$. By definition of Y, for each $y^{(j)}$ with $1 \leq j \leq |Y|$ there exists some index i_j with $y_{i_j}^{(j)} = 1$. Because we additionally have $\forall \ell : 1 \leq \ell < j,\ y_{i_\ell}^{(j)} = 0$, all such indices i_j are different. An n-bit bitstring cannot have more than n different indices and hence an upper bound for $|Y|$ is n. Hence, the circuit depth for $\mathcal{C}_Y$ is at most quadratic in n. □

CHAPTER 7

Removing the Zero Vector from a Superposition

In the previous chapter, we discussed a method to remove any non-zero bitstring from a superposition. When we inspect the superposition produced by Algorithm 5.1, we realize however, that this is not enough. For a hidden subgroup H, the superposition at the end of Theorem 3 is over elements of $H^\perp$ and by construction, we always have $\mathbf{0} \in H^\perp$. This is a problem, since in the setting of the extended version of Simon's problem, we are looking for linearly independent elements from $H^\perp$. The zero vector is never part of a linearly independent set of vectors, hence if we measure it, we do not get additional information. The possibility to repeatedly measure the zero vector also renders the procedure developed so far a probabilistic one, but we are interested in a deterministic algorithm with a fixed runtime. Consequently, we need an additional modification to the original version of Simon's algorithm which removes the zero vector from superpositions over $H^\perp$.

We first present quantum operators and their implementation, which serve as building blocks for a larger quantum algorithm that removes the zero vector from a superposition. Second, following Lemma 8 from Brassard and Høyer [13], we state such a quantum algorithm itself and prove its correctness and computational complexity.

7.1 Preparatory Quantum Operators

The following technical result helps us with the implementation of complex quantum operators. Intuitively the lemma states that, in order to shift the global phase of a quantum register $|\psi\rangle$ by i, it is enough to just perform the shift locally on one qubit only.

Lemma 14 (Phase Shift Lemma)**.** *Let $|\psi\rangle = |q_{n-1}q_{n-2}\dots q_0\rangle$ be an n-qubit quantum state with $n > 0$. Then for an arbitrary j with $0 \leq j < n$ we have*

$$\left(\mathcal{I}^{\otimes j} \otimes (i \cdot \mathcal{I}) \otimes \mathcal{I}^{\otimes n-j-1}\right) |\psi\rangle = i\,|\psi\rangle\,.$$

Proof. Let $|\psi\rangle = |q_{n-1}q_{n-2}\dots q_0\rangle$ be an n-qubit register. Fix an arbitrary qubit $|q_j\rangle$, $0 \leq j < n$. We examine the operator that changes the phase of $|q_j\rangle$ only and does not manipulate any other qubit. In the following, note that the edge cases $j = 0$ and $j = n-1$ are fully covered because we defined $\mathcal{I}^{\otimes 0} = (1)$.

$$\mathcal{I}^{\otimes j} \otimes (i \cdot \mathcal{I}) \otimes \mathcal{I}^{\otimes n-j-1} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{2^j \times 2^j} \otimes \begin{pmatrix} i & 0 \\ 0 & i \end{pmatrix} \otimes \mathcal{I}^{\otimes n-j-1}$$

$$= \underbrace{\begin{pmatrix} i & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & i \end{pmatrix}}_{2^{j+1} \times 2^{j+1}} \otimes \mathcal{I}^{\otimes n-j-1} = i \cdot \mathcal{I}^{\otimes j+1} \otimes \mathcal{I}^{\otimes n-j-1} = i \cdot \mathcal{I}^n.$$

Hence, $\left(\mathcal{I}^{\otimes j} \otimes (i \cdot \mathcal{I}) \otimes \mathcal{I}^{\otimes n-j-1}\right) |\psi\rangle = i \cdot |\psi\rangle$. □

Next, we introduce non-standard quantum operators and specify procedures for how to implement them as quantum circuits.

Lemma 15 (Lemma $\mathcal{S}_{\{0\}}$)**.** *Let $|x\rangle \in \mathfrak{B}_n$ be a state from the n-qubit computational standard basis. Let $\mathcal{S}_{\{0\}} = \mathcal{I}^{\otimes n} - |\mathbf{0}\rangle\langle\mathbf{0}| + i|\mathbf{0}\rangle\langle\mathbf{0}|$. Then we have*

$$\mathcal{S}_{\{0\}} |x\rangle = \begin{cases} i\,|x\rangle & |x\rangle = |\mathbf{0}\rangle\,, \\ |x\rangle & \textit{otherwise.} \end{cases}$$

Proof. Fix an arbitrary $|x\rangle \in \mathfrak{B}_n$. Then we have

$$\begin{aligned} \mathcal{S}_{\{0\}} |x\rangle &= \left(\mathcal{I}^{\otimes n} - |\mathbf{0}\rangle\langle\mathbf{0}| + i|\mathbf{0}\rangle\langle\mathbf{0}|\right) |x\rangle \\ &= |x\rangle - |\mathbf{0}\rangle \langle\mathbf{0}|x\rangle + i\,|\mathbf{0}\rangle \langle\mathbf{0}|x\rangle\,. \end{aligned}$$

Now we perform a case distinction on $|x\rangle$.

Case $|x\rangle = |\mathbf{0}\rangle$. The above expression $\mathcal{S}_{\{0\}} |x\rangle$ simplifies to

$$|\mathbf{0}\rangle - |\mathbf{0}\rangle \langle\mathbf{0}|\mathbf{0}\rangle + i\,|\mathbf{0}\rangle \langle\mathbf{0}|\mathbf{0}\rangle = i\,|\mathbf{0}\rangle\,,$$

where in the last step we use that $\langle\mathbf{0}|\mathbf{0}\rangle = 1$, because $|\mathbf{0}\rangle$ is a computational basis state.

Case $|x\rangle \neq |\mathbf{0}\rangle$. We use the fact that $|x\rangle$ and $|\mathbf{0}\rangle$ are distinct states from the orthogonal computational basis and hence $\langle \mathbf{0}|x\rangle = 0$. Consequently, the above expression $\mathcal{S}_{\{0\}}|x\rangle$ simplifies to $|x\rangle$.

Thus, we have that $\mathcal{S}_{\{0\}}$ changes the phase of the input state $|x\rangle$ by the imaginary unit i if and only if $|x\rangle = |\mathbf{0}\rangle$. □

We also specify how to implement the $\mathcal{S}_{\{0\}}$ operator as a quantum circuit.

Lemma 16. *Let $\mathcal{S}_{\{0\}}$ be the quantum operator described in Lemma 15, operating on an n-qubit register. Then $\mathcal{S}_{\{0\}}$ can be implemented as a quantum circuit of depth linear in n.*

Proof. We first consider the special case $n = 1$. The corresponding circuit is shown in Figure 7.1. If the input qubit is in state $|0\rangle$, then the first X gate will put it in state $|1\rangle$. Then the S gate performs the phase shift, and we reset the input qubit back to $|0\rangle$ with another X gate. If the input qubit is in state $|1\rangle$, then the first X gate puts it into state $|0\rangle$, where the S gate has no effect and hence no phase shift is performed. The second X gate again resets the input qubit to its original state.

Next, we consider the special case $n = 2$. The corresponding circuit is also shown in Figure 7.1. Note that we now assume an ancillary qubit in state $|0\rangle$. Recall that by the Phase Shift Lemma (Lemma 14), if we want to shift the phase of an entire register, it is enough to just perform the phase shift on any one of its qubits. Hence, for us, it is enough to apply the S gate to the ancillary qubit. The rest of the analysis remains analogous to the previous case.

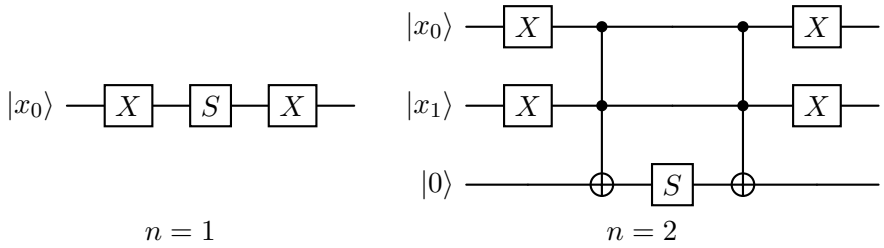


Figure 7.1: $\mathcal{S}_{\{0\}}$ for $n \in \{1, 2\}$, least significant qubit is topmost

The cases where $n > 2$ work analogously. However, multi-controlled X gates are non-standard, and we have to simulate them. One possible simulation method is given by Nielsen and Chuang in Section 4.3 of [27]. We show an example circuit for $n = 5$ and with 4 ancillary qubits in Figure 7.2.

For a working register of size n, we first apply n X gates in parallel, adding to our circuit depth 1. Next we execute $(n-1)$ *CCNOT* gates sequentially, which increases the depth of the circuit by $n - 1$. After that, we apply one S gate and undo the previous steps by running them in reverse. In total, we thus have a circuit depth of $1 + (n-1) + 1 + (n-1) + 1 = 2n + 1$, which is linear in n. □

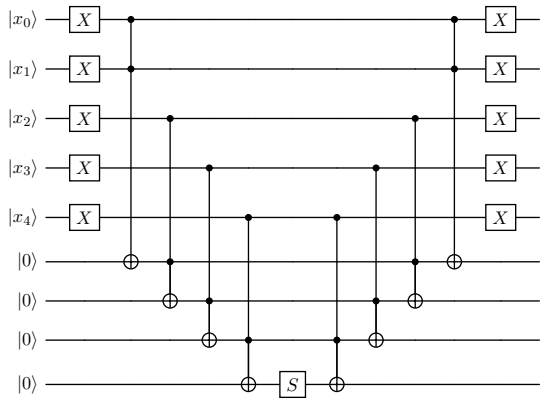


Figure 7.2: $\mathcal{S}_{\{0\}}$ for $n = 5$, least significant qubit is topmost

The upside of the proposed circuit for $\mathcal{S}_{\{0\}}$ is that its depth grows only linearly with the size of the working quantum register. However, the downside is that we need $n - 1$ ancillary qubits and those are scarce on NISQ hardware. As noted by Barenco et al. [8], multi-controlled X gates can also be simulated entirely without ancillary qubits. We do not follow that approach because then the circuit depth would grow exponentially in n.

7.2 Quantum Algorithm for Removing the Zero Vector

We now provide a quantum algorithm to remove the zero state $|\mathbf{0}\rangle$ from a superposition, closely following Lemma 8 from [13], but making some adjustments for simplicity and clarity. The algorithm uses a particular form of a technique called *amplitude amplification* [14, 18], where intuitively we are given a superposition over 'good' states which we would like to measure and 'bad' states which we would not like to measure. Our goal is to increase the amplitudes of the good states and to decrease the amplitudes of the bad states. In our particular case, we can even boost the amplitude of the good states such that the probability to measure one of them is 100%.

The setting is as follows. We are working on one quantum register of length n (not counting ancillary qubits). Let $X \subseteq G$ be some subset of group G (but not necessarily a subgroup). Let $\mathcal{A}$ be a measurement-free quantum algorithm composed of self-adjoint operators that performs the state transition $|\mathbf{0}\rangle \to |\Psi\rangle$ where

$$|\Psi\rangle = \sum_{x \in X} \alpha_x |x\rangle, \quad \forall x \in X : \alpha_x \in \mathbb{C}, \quad \langle \Psi | \Psi \rangle = 1.$$

At this point we do not care about the internal construction of $\mathcal{A}$, we only care about its effects on $|\mathbf{0}\rangle$. We will instantiate $\mathcal{A}$ with a concrete quantum algorithm later in Section 8.2.

Let $\chi : X \to \{0, 1\}$ be a total Boolean function. We use χ to partition the set X into

two disjoint subsets

$$A = \{x \in X \mid \chi(x) = 1\} \text{ and } B = \{x \in X \mid \chi(x) = 0\},$$

where intuitively A is the set of good states and B is the set of bad states. We note that by construction, $A \cap B = \emptyset$ and $A \cup B = X$. Hence, we can write the superposition $|\Psi\rangle$ as $|A\rangle + |B\rangle$ where

$$|A\rangle = \sum_{x \in A} \alpha_x |x\rangle \text{ and } |B\rangle = \sum_{x \in B} \alpha_x |x\rangle .$$

We recall from Section 2.3 that for each $x \in X$, a quantum register $|x\rangle$ that holds the value x, is in a state from the orthogonal computational basis of $\mathbb{C}^{2^n}$. Consequently, when a register is in state $|\Psi\rangle$, the probability to measure a state from $|A\rangle$ or $|B\rangle$ is $a = \langle A|A\rangle$ or $b = \langle B|B\rangle$ respectively.

Additionally, we assume that we are given an $n+1$ qubit quantum operator $\mathcal{U}_\chi$, which implements χ. That is, we have $\mathcal{U}_\chi |x\rangle |q\rangle = |x\rangle |q \oplus \chi(x)\rangle$. In particular, $\mathcal{U}_\chi |x\rangle |0\rangle = |x\rangle |1\rangle$ for $x \in A$ and $\mathcal{U}_\chi |x\rangle |0\rangle = |x\rangle |0\rangle$ for $x \in B$. Note that $|q\rangle$ is merely an ancillary qubit used in the implementation of $\mathcal{U}_\chi$. It is omitted in the following mathematical analysis.

The deviations from the definition by Brassard and Høyer [13] are as follows. We explicitly state that the function χ must be total. Furthermore, in the original formulation, the authors use the symbol I for the set here denoted by X. Members of the set I are originally denoted with the i symbol rather than with the x symbol here. The symbol i is already reserved for the imaginary unit, hence the renaming. Moreover, in their original formulation, the authors do not explicitly state the amplitudes α_x for the elements in the superposition in $|\Psi\rangle$. Instead, they introduce a second quantum register that holds non-basis states entangled with the first register in order to ensure that $|\Psi\rangle$ is a valid quantum state. Introducing a second register makes the notation for this section even more complicated than it already is, hence the rework.

The original statement of Lemma 8 has two parts:

(A) There exists a quantum algorithm $\mathcal{Q}$ that brings us from the state $|\mathbf{0}\rangle$ to the state

$$(2i(1-a) - 1) |A\rangle + i(1-2a) |B\rangle .$$

(B) If $\mathcal{A}$ and $\mathcal{U}_\chi$ use no measurement operators, then $\mathcal{Q}$ uses no measurement operators as well. Furthermore, the circuit depth of $\mathcal{Q}$ is linear in the quantum register size n and constant in the circuit depths of $\mathcal{A}$ and $\mathcal{U}_\chi$.

In the paper by Brassard and Høyer [13], there is only one high-level proof that covers both statements. Here, we give two separate theorems with separate, detailed proofs.

Theorem 5 (Remove Zero A)**.** *There exists a quantum algorithm $\mathcal{Q}$ that on input $|\mathbf{0}\rangle$ returns*

$$(2i(1-a)-1)\,|A\rangle + i(1-2a)\,|B\rangle\,,$$

where $a = \langle A|A\rangle$.

Proof. Following the presentation of Brassard and Høyer [13], we note the following:

- The state $|\Psi\rangle$ is a superposition of some states from the computational standard basis $\mathfrak{B}_n$. By definition, all its states are orthogonal to each other. That means we have $\langle A|B\rangle = \langle B|A\rangle = 0$.
- Because $A \cap B = \emptyset$ and $A \cup B = X$, we have $a + b = 1$ and in particular $b = 1 - a$.

We recall the definition of the quantum operator $\mathcal{S}_{\{0\}}$ from Lemma 15 and introduce the new quantum operator $\mathcal{S}_A$:

$$\mathcal{S}_{\{0\}}\,|x\rangle = \begin{cases} i\,|x\rangle & x = |\mathbf{0}\rangle\,, \\ |x\rangle & \text{otherwise.} \end{cases} \qquad \mathcal{S}_A\,|x\rangle = \begin{cases} i\,|x\rangle & x \in A, \\ |x\rangle & \text{otherwise.} \end{cases}$$

Claim: Let $G = \mathcal{A}\mathcal{S}_{\{0\}}\mathcal{A}^{-1}\mathcal{S}_A$. Then $\mathcal{Q}$ is realized by the operator $G\mathcal{A}$. What follows is the analysis of what happens when we apply $\mathcal{Q}$ to the initial state $|\mathbf{0}\rangle$.

We recall that for $\mathcal{A}$ we have $\mathcal{A}\,|\mathbf{0}\rangle = |\Psi\rangle$. Hence, for $\mathcal{Q}$ we have

$$\mathcal{Q}\,|\mathbf{0}\rangle = G\mathcal{A}\,|\mathbf{0}\rangle = G\,|\Psi\rangle = G(|A\rangle + |B\rangle) = G\,|A\rangle + G\,|B\rangle\,.$$

For the further steps we go into detail about how G is implemented. We make some observations.

- The effects of $\mathcal{S}_A$ on $|A\rangle$ and $|B\rangle$ can be described as follows:

$$\mathcal{S}_A\,|A\rangle = \mathcal{S}_A \sum_{x\in A} \alpha_x\,|x\rangle = \sum_{x\in A} \alpha_x \mathcal{S}_A\,|x\rangle = i \sum_{x\in A} \alpha_x\,|x\rangle = i\,|A\rangle$$
$$\mathcal{S}_A\,|B\rangle = \mathcal{S}_A \sum_{x\in B} \alpha_x\,|x\rangle = \sum_{x\in B} \alpha_x \mathcal{S}_A\,|x\rangle = \sum_{x\in B} \alpha_x\,|x\rangle = |B\rangle$$

- By invoking Lemma 15, we can express $\mathcal{S}_{\{0\}}$ as

$$\mathcal{S}_{\{0\}} = \mathcal{I} - |\mathbf{0}\rangle\langle\mathbf{0}| + i|\mathbf{0}\rangle\langle\mathbf{0}|.$$

- Recall $\mathcal{A}\,|\mathbf{0}\rangle = |\Psi\rangle = |A\rangle + |B\rangle$ and that $\mathcal{A}$ is unitary by assumption. This allows us to derive a helpful identity.

$$\begin{aligned} \mathcal{A}\,|\mathbf{0}\rangle\,\langle\mathbf{0}|\,\mathcal{A}^{-1} = |\Psi\rangle\langle\Psi| &= (\,|A\rangle + |B\rangle\,)(\,\langle A| + \langle B|\,) \\ &= |A\rangle\langle A| + |A\rangle\langle B| + |B\rangle\langle A| + |B\rangle\langle B| \end{aligned}$$

Now we first calculate $G\,|A\rangle$.

$$\begin{aligned} G\,|A\rangle &= \mathcal{A}\mathcal{S}_{\{0\}}\mathcal{A}^{-1}\mathcal{S}_A\,|A\rangle = \mathcal{A}\mathcal{S}_{\{0\}}\mathcal{A}^{-1}i\,|A\rangle = \mathcal{A}(\mathcal{I}-|\mathbf{0}\rangle\langle\mathbf{0}|+i|\mathbf{0}\rangle\langle\mathbf{0}|)\mathcal{A}^{-1}i\,|A\rangle \\ &= (\mathcal{A}\mathcal{A}^{-1}-\mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1}+i\mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1})\;\; i\,|A\rangle \\ &= (\mathcal{I}-\mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1}+i\mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1})\;\; i\,|A\rangle \\ &= i\,|A\rangle - \mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1}i\,|A\rangle + i\mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1}i\,|A\rangle \\ &= i\,|A\rangle - \mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1}i\,|A\rangle - \mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1}\,|A\rangle \end{aligned}$$

We analyze the term $\mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1}\,i\,|A\rangle$.

$$\begin{aligned} \mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1}\;\; i\,|A\rangle &= \left(|A\rangle\langle A| + |A\rangle\langle B| + |B\rangle\langle A| + |B\rangle\langle B|\right)\;\; i\,|A\rangle \\ &= i\,|A\rangle\,\langle A|A\rangle + i\,|A\rangle\,\langle B|A\rangle + i\,|B\rangle\,\langle A|A\rangle + i\,|B\rangle\,\langle B|A\rangle \\ &= ia\,|A\rangle + ia\,|B\rangle \end{aligned}$$

In the same style, we derive

$$\mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1}\;\; |A\rangle = a\,|A\rangle + a\,|B\rangle\,.$$

The two identities just derived help us to simplify $G\,|A\rangle$ to

$$G\,|A\rangle = i\,|A\rangle - ia\,|A\rangle - ia\,|B\rangle - a\,|A\rangle - a\,|B\rangle\,.$$

Next we consider $G\,|B\rangle$.

$$\begin{aligned} G\,|B\rangle &= \mathcal{A}\mathcal{S}_{\{0\}}\mathcal{A}^{-1}\mathcal{S}_A\,|B\rangle = \mathcal{A}\mathcal{S}_{\{0\}}\mathcal{A}^{-1}\,|B\rangle = \mathcal{A}(\mathcal{I}-|\mathbf{0}\rangle\langle\mathbf{0}|+i|\mathbf{0}\rangle\langle\mathbf{0}|)\mathcal{A}^{-1}\,|B\rangle \\ &= (\mathcal{I}-\mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1}+i\mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1})\;\; |B\rangle \\ &= |B\rangle - \mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1}\,|B\rangle + i\mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1}\,|B\rangle \end{aligned}$$

We analyze the term $\mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1}\,|B\rangle$, where in the last step we use $b = 1-a$.

$$\begin{aligned} \mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1}\;\; |B\rangle &= \left(|A\rangle\langle A| + |A\rangle\langle B| + |B\rangle\langle A| + |B\rangle\langle B|\right)\;\; |B\rangle \\ &= |A\rangle\,\langle A|B\rangle + |A\rangle\,\langle B|B\rangle + |B\rangle\,\langle A|B\rangle + |B\rangle\,\langle B|B\rangle \\ &= b\,|A\rangle + b\,|B\rangle \\ &= |A\rangle - a\,|A\rangle + |B\rangle - a\,|B\rangle \end{aligned}$$

We immediately get

$$i\mathcal{A}|\mathbf{0}\rangle\langle\mathbf{0}|\mathcal{A}^{-1}\;\; |B\rangle = i\,|A\rangle - ia\,|A\rangle + i\,|B\rangle - ia\,|B\rangle\,.$$

The two identities just derived help us to simplify $G\,|B\rangle$.

$$G\,|B\rangle = |B\rangle - |A\rangle + a\,|A\rangle - |B\rangle + a\,|B\rangle + i\,|A\rangle - ia\,|A\rangle + i\,|B\rangle - ia\,|B\rangle$$

Finally, we combine the two results.

$$\begin{aligned} G|A\rangle + G|B\rangle &= i|A\rangle - ia|A\rangle - ia|B\rangle - a|A\rangle - a|B\rangle + \\ &\quad |B\rangle - |A\rangle + a|A\rangle - |B\rangle + a|B\rangle + i|A\rangle - ia|A\rangle + i|B\rangle - ia|B\rangle \\ &= 2 \cdot i|A\rangle - 2 \cdot ia|A\rangle - |A\rangle - 2 \cdot ia|B\rangle + i|B\rangle \\ &= (2i(1-a) - 1)|A\rangle + i(1-2a)|B\rangle \end{aligned}$$

□

Theorem 6 (Remove Zero B)**.** *If $\mathcal{A}$ and $\mathcal{U}_\chi$ use no measurement operators, then $\mathcal{Q}$ uses no measurement operators. Furthermore, the circuit depth of $\mathcal{Q}$ is linear in the quantum register size n and constant in the circuit depths of $\mathcal{A}$ and $\mathcal{U}_\chi$.*

Proof. Assume that $\mathcal{A}$ uses no measurements. Recalling the definition of $\mathcal{Q}$, we have $\mathcal{Q} = G\mathcal{A}$ and $G = \mathcal{A}\mathcal{S}_{\{0\}}\mathcal{A}^{-1}\mathcal{S}_A$. In $\mathcal{Q}$ we have two applications of $\mathcal{A}$ and one application of $\mathcal{A}^{-1}$, which in total contribute three times the circuit depth of $\mathcal{A}$ to the circuit depth of $\mathcal{Q}$.

The operator $\mathcal{S}_{\{0\}}$, by Lemma 16, can be implemented as a circuit without measurement operations that has depth linear in n. We still have to show that the last building block of $\mathcal{Q}$, the operator $\mathcal{S}_A$ can be implemented as a circuit with depth linear in n and constant in the depth of the implementation of $\mathcal{U}_\chi$. In the following, we present such an implementation. Assume we are working on two quantum registers, the first one of size n and the second one, holding only an ancillary qubit *aux*, of size 1. Let $|\psi_0\rangle = |x\rangle|0\rangle$, where $|x\rangle$ is any state from the superposition $|\Psi\rangle$. Then, the following quantum algorithm, which directly translates into a circuit (shown in Figure 7.3), implements $\mathcal{S}_A$ on $|\psi_0\rangle$:

1. $|\psi_1\rangle = \mathcal{U}_\chi|\psi_0\rangle$ — *aux* is $|1\rangle$ iff $x \in A$
2. $|\psi_2\rangle = (\mathcal{I}^{\otimes n} \otimes \mathcal{S})|\psi_1\rangle$ — perform conditional phase shift on *aux*
3. $|\psi_3\rangle = \mathcal{U}_\chi|\psi_2\rangle$ — reset *aux* qubit

Suppose $x \in A$. Then, after step 1., the ancillary qubit is in state $|1\rangle$. In step 2., we apply the $\mathcal{S}$ operator to the ancillary qubit. By the Phase Shift Lemma (Lemma 14), this puts the entire register in state

$$|\psi_2\rangle = i \cdot \Big(|x\rangle|1\rangle\Big).$$

Finally, we apply $\mathcal{U}_\chi$ again, resetting the ancillary qubit and putting the register in state

$$|\psi_3\rangle = i \cdot \Big(|x\rangle|0\rangle\Big) = i \cdot |\psi_0\rangle.$$

Suppose $x \notin A$. Then, after step 1., the ancillary qubit is still in state $|0\rangle$. Thus, applying the $\mathcal{S}$ operator in step 2. has no effect and neither does the application of $\mathcal{U}_\chi$ in step 3.

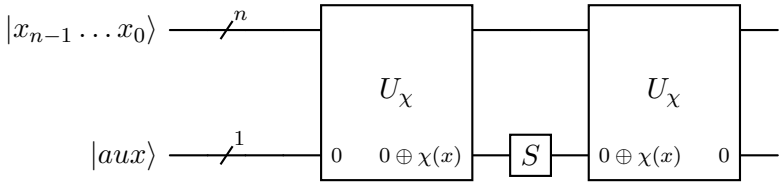


Figure 7.3: $\mathcal{S}_A$ circuit, U_χ transforms aux only, least significant qubit is topmost

With the S gate being standard (no expensive simulation required), the construction step 2. adds only one gate to the $\mathcal{S}_A$ circuit. Steps 1. and 3. require a total amount of two $\mathcal{U}_\chi$ applications, so the depth of the circuit of $\mathcal{S}_A$ is constant in the depth of the circuit of $\mathcal{U}_\chi$. As a consequence, the circuit for $\mathcal{Q}$ meets the conditions from the statement of the theorem. □

CHAPTER 8

A Deterministic Algorithm for the Extended Version of Simon's Problem

In the previous chapters 5, 6 and 7 we collected the building blocks for an algorithm that deterministically solves the extended version of Simon's problem. We still need to piece them together and give a formal description of the algorithm, which we are going to do in this chapter. First, we prove some preparatory results needed for the orchestration of the individual building blocks. Next, we state and (using the results from the previous chapters) prove Theorem 4 from Brassard and Høyer [13], in which they present an enhanced version of the quantum routine by Simon. This routine always outputs a new piece of information (i.e. a previously unknown element of $H^\perp$). Following the presentation of Brassard and Høyer in their Theorem 5 [13], we then present the overall quantum routine that recovers a generating set of $H^\perp$. Finally, we present an example run of the new algorithm.

8.1 Preparatory Results

In our introduction, we defined the orthogonal set of a subgroup of G. We prove now that the orthogonal set is even a group.

Lemma 17 (Orthogonal Group Lemma)**.** *Let H be a subgroup of the group $G = \langle \{0,1\}^n, \oplus \rangle$. Then $\langle H^\perp, \oplus \rangle$ is a group.*

Proof. It is straightforward to see that $H^\perp$ is non-empty. First we note that $\mathbf{0} \in G$. Then, for arbitrary $h \in H$ we have

$$\mathbf{0} \odot h = 0 \cdot h_{n-1} \oplus \cdots \oplus 0 \cdot h_0 = 0,$$

and hence $\mathbf{0} \in H^\perp$. It is only left to show that $\langle H^\perp, \oplus \rangle$ satisfies the group axioms:

1. **Closure**: Fix arbitrary elements $h \in H$ and $g_1, g_2 \in H^\perp$. Then, by the Distributivity Lemma (Lemma 1) we have $h \odot (g_1 \oplus g_2) = (h \odot g_1) \oplus (h \odot g_2)$. Because $g_1, g_2 \in H^\perp$, we have $h \odot g_1 = h \odot g_2 = 0$. By definition of $\oplus$, $0 \oplus 0 = 0$ and thus $(g_1 \oplus g_2) \in H^\perp$.
2. **Neutral Element**: The neutral element of $H^\perp$ is $\mathbf{0}$ since for arbitrary $h \in H^\perp$ we have $h \oplus \mathbf{0} = h$.
3. **Inverse Element**: By definition of $\oplus$, each $g \in H^\perp$ is its own inverse.
4. **Associativity**: Follows directly from the definition of $\oplus$.

□

When we want to recover a hidden subgroup H, our goal is to collect a generating set for $H^\perp$, because we can turn this into a generating set for H. With H being hidden, we do not know $|H|$, and thus we also do not know $|H^\perp|$. This causes a problem: When we construct a basis for $H^\perp$ by collecting linearly independent elements from $H^\perp$, how do we know when we are done? The following lemma answers that question.

Lemma 18 (K-Y-H Lemma)**.** *Let H be a subgroup of the group $G = \langle \{0,1\}^n, \oplus \rangle$ and let $Y = \{y^{(1)}, \ldots, y^{(|Y|)}\} \subseteq H$ be a set where each $y^{(j)} \in Y$ satisfies the following two formulas*

- $\exists i_j : y^{(j)}_{i_j} = 1$, *and*
- $\forall \ell : 1 \leq \ell < j,\ y^{(j)}_{i_\ell} = 0$.

Let $K = \{h \in H \mid h_{i_1} = h_{i_2} = \cdots = h_{i_{|Y|}} = 0\}$. Then we have

$$\langle Y \rangle = H \iff K = \{\mathbf{0}\}$$

Proof, $\Longrightarrow$. We prove $\langle Y \rangle = H \implies K = \{\mathbf{0}\}$. Assume towards a contradiction that $\langle Y \rangle = H$ but $K \neq \{\mathbf{0}\}$. Note that $\mathbf{0} \in H$ for arbitrary H and thus by construction of K, we always have $\{\mathbf{0}\} \subseteq K$. From the assumption $K \neq \{\mathbf{0}\}$, we conclude that there exists some bitstring x, $x \neq \mathbf{0}$ such that $\{\mathbf{0}, x\} \subseteq K$. The fact that $x \in K$ gives us two important pieces of information:

(F1) By the construction of K we get that $x_{i_1} = x_{i_2} = \cdots = x_{i_{|Y|}} = 0$.

(F2) Since $x \in K$ and $K \subseteq H$, we get $x \in H$ and by our assumption $\langle Y \rangle = H$ we get $x \in \langle Y \rangle$.

Because of (F2), there must exist some subset Y' of Y with $Y' = \{y^{(a_1)}, \ldots, y^{(a_m)}\}$, such that $x = y^{(a_1)} \oplus \cdots \oplus y^{(a_m)}$. Note that $|Y'| \geq 1$, since $x \neq \mathbf{0}$ and an empty set only generates $\{\mathbf{0}\}$. In other words, there must exist at least one element of $Y = \{y^{(1)}, \ldots, y^{(|Y|)}\}$ which is also contained in Y'. We will bring this to a contradiction by proving by strong mathematical induction on the elements of Y that no such element is also contained in Y'.

Induction Start We prove $y^{(1)} \notin Y'$ indirectly, so we assume towards a contradiction that $y^{(1)} \in Y'$. Let i_1 be such that $y^{(1)}_{i_1} = 1$. Then by construction of Y, we must have that for all elements of Y' except $y^{(1)}$, their bit value at index i_1 is 0 and thus $(y^{(a_1)} \oplus \cdots \oplus y^{(a_m)})_{i_1} = 1$. This is a contradiction to $x_{i_1} = 0$, which we derived in (F1).

Induction Hypothesis Let $j > 1$ and assume that for all $\ell, 1 \leq \ell < j$, we have $y^{(\ell)} \notin Y'$.

Induction Step Consider $y^{(j)}$ with $y^{(j)}_{i_j} = 1$ and assume towards a contradiction that $y^{(j)} \in Y'$. By construction of Y we have that for all ℓ with $1 \leq \ell < j$, $y^{(j)}_{i_\ell} = 0$ and, by hypothesis, $y^{(\ell)} \notin Y'$. Hence, all elements in Y that still can be in Y' are of form $y^{(k)}$ with $k \geq j$. Since by construction of Y those elements must all satisfy $y^{(k)}_{i_j} = 0$, we must have $x_{i_j} = 1$, which is a contradiction to $x_{i_j} = 0$, which we derived before in (F1).

We derived both that Y' is a non-empty subset of Y and that no element of Y' is also an element of Y, which is a contradiction. The only assumption we made so far is that $K \neq \{\mathbf{0}\}$, so the opposite must be true, and we must have $K = \{\mathbf{0}\}$. □

Proof, $\Longleftarrow$. We prove $K = \{\mathbf{0}\} \implies \langle Y \rangle = H$. Let $K = \{\mathbf{0}\}$ and assume towards a contradiction that $\langle Y \rangle \neq H$. By our assumption, we get that there exists some $h \in H$ such that $h \notin \langle Y \rangle$. We define $K_{(0)} = H$, and for each j with $1 \leq j \leq |Y|$ we define $K_{(j)} = \{h \in H \mid h_{i_1} = \cdots = h_{i_j} = 0\}$. Then the following statement holds:

For each j with $0 \leq j \leq |Y|$ there exists a bitstring $k^{(j)} \in K_{(j)}$ and a sequence of bitstrings $y^{(\ell_1)}, \ldots, y^{(\ell_m)}$ from Y with $0 \leq m \leq j$ (if $m = 0$, then the sequence is empty) such that $h = k^{(j)} \oplus y^{(\ell_1)} \oplus \cdots \oplus y^{(\ell_m)}$.

We prove this statement via mathematical induction on j.

Induction Start $j = 0$. We need to show that $h = k^{(0)}$ for some $k^{(0)} \in K_{(0)}$. By definition, $K_{(0)} = H$ and we know that $h \in H$. Hence, we can set $k^{(0)} = h$, and we are done.

Induction Hypothesis For some j with $0 \leq j \leq |Y|$ there exist a bitstring $k^{(j)} \in K_{(j)}$ and a sequence of bitstrings $y^{(\ell_1)}, \ldots, y^{(\ell_m)}$ from Y with $m \leq j$ such that $h = k^{(j)} \oplus y^{(\ell_1)} \oplus \cdots \oplus y^{(\ell_m)}$.

Induction Step By induction hypothesis we can write $h = k^{(j)} \oplus y^{(\ell_1)} \oplus \cdots \oplus y^{(\ell_m)}$. Consider first the case that $k^{(j)} = \mathbf{0}$ ($\mathbf{0}$ holds a 0 in each bit). By definition, $\mathbf{0} \in K_{(j+1)}$. Furthermore, since $m \leq j$ we also have $m \leq j + 1$, and as a consequence, for $j + 1$, we can still write $h = k^{(j)} \oplus y^{(\ell_1)} \oplus \cdots \oplus y^{(\ell_m)}$.

Consider next the case that $k^{(j)} \neq \mathbf{0}$. Because $K = \{\mathbf{0}\}$, we have that $k^{(j)} \notin K$ and thus there must exist some a with $a > j$ such that $k^{(j)} \in K_{(a-1)}$ but $k^{(j)} \notin K_{(a)}$. By Corollary 2 from Section 6.1, we can thus write $k^{(j)} = k' \oplus y^{(a)}$ for some $k' \in K_{(a)}$. We insert this equality to the construction of h from the hypothesis, $h = k^{(j)} \oplus y^{(\ell_1)} \oplus \cdots \oplus y^{(\ell_m)}$, and get

$$h = k' \oplus \overbrace{\underbrace{y^{(\ell_1)} \oplus \cdots \oplus y^{(\ell_m)}}_{\leq j \text{ elements}} \oplus y^{(a)}}^{\leq j+1 \text{ elements}},$$

where we recall that $k' \in K_{(a)}$ and because $a > j$ we must also have that $k' \in K_{(j+1)}$, which concludes the induction step from j to $j+1$.

We instantiate the statement just proven with $j = |Y|$, which gives us

$$h = k^{(|Y|)} \oplus y^{(\ell_1)} \oplus \cdots \oplus y^{(\ell_m)}$$

for some $m \leq |Y|$. Since $k^{(|Y|)} \in K_{(i_{|Y|})} = K = \{\mathbf{0}\}$, we infer that $k^{(|Y|)} = \mathbf{0}$ and hence we can write $h = y^{(\ell_1)} \oplus \cdots \oplus y^{(\ell_m)}$, which is a contradiction to the assumption that $h \notin \langle Y \rangle$. □

8.2 Deterministic Quantum Algorithm for the Extended Version of Simon's Problem

The next theorem was originally stated by Brassard and Høyer [13] as Theorem 4. In their original version, Y is just a linearly independent set. We strengthen that condition to simplify the mathematical analysis.

Theorem 7. *Let H be any subgroup of the group $G = \langle \{0,1\}^n, \oplus \rangle, n > 1$ and let $\rho : G \to R$ be a function that fulfills Simon's promise and where R is some set representable on an m-qubit quantum register. Assume that a quantum algorithm $\mathcal{U}_\rho$ that computes ρ without making any measurement is given, together with the value of n and an arbitrary subset $Y \subseteq H^\perp$ where each element $y^{(j)} \in Y$ satisfies the following two formulas*

- $\exists i_j : y^{(j)}_{i_j} = 1$, *and*
- $\forall \ell : 1 \leq \ell < j, \;\; y^{(j)}_{i_\ell} = 0.$

Then there exists a quantum algorithm that returns an element of $H^\perp \setminus \langle Y \rangle$ provided Y does not generate $H^\perp$, and otherwise it returns the zero element. Moreover, the algorithm can be implemented as a quantum circuit of depth at most n times the circuit depth of $\mathcal{U}_\rho$ plus a number of other quantum computation steps within $O(n^3)$.

Proof. We are working on three quantum registers of size n, m and $|Y|$. For better readability, any ancillary qubits needed for the computation of $\mathcal{U}_\rho$ or multi-controlled gates are omitted. Initially, all qubits are set to the $|0\rangle$ state, and we have

$$|\psi_0\rangle = |0\rangle^{\otimes n} |0\rangle^{\otimes m} |0\rangle^{\otimes |Y|}.$$

Let $\mathcal{B}$ denote Algorithm 5.1, but where we skip the measurement step at the very end. Then by Theorem 3 we get

$$|\psi_1\rangle = \mathcal{B}\mathcal{I}^{\otimes |Y|} |\psi_0\rangle = \frac{1}{\sqrt{|T|}} \Big(\sum_{t \in T} \left|\phi_t H^\perp\right\rangle |\rho(t)\rangle \Big) |0\rangle^{\otimes |Y|}.$$

In the original version of Simon's algorithm we would not skip measuring the first register now, and we would obtain some random $y \in H^\perp$. In particular, it could happen that we measure some $y \in \langle Y \rangle$, i.e. some element that we already discovered and that is of no use. To avoid repeated measurements, we note that by the Orthogonal Group Lemma (Lemma 17), $H^\perp$ is a group, hence we can apply the quantum algorithm $\mathcal{C}_Y$ from the Blocking Clause Theorem (Theorem 4) to the state $|\psi_1\rangle$. Note that $\mathcal{C}_Y$ operates on two registers, but in state $|\psi_1\rangle$ we have three. We choose the first register from $|\psi_1\rangle$ as the first register for $\mathcal{C}_Y$ and the last register from $|\psi_1\rangle$ as the second register for $\mathcal{C}_Y$. That is, the application of $\mathcal{C}_Y$ does not change the state of the middle register in $|\psi_1\rangle$. The result is

$$|\psi_2\rangle = \mathcal{C}_Y |\psi_1\rangle = \frac{1}{\sqrt{|T|}} \Big(\left|\phi_t K_{(|Y|)}\right\rangle |\rho(t)\rangle \Big(\bigotimes_{y^{(j)} \in Y} \left|t \odot y^{(j)}\right\rangle \Big) \Big).$$

By definition, each $y^{(j)} \in Y$ has an index i_j for $1 \leq j \leq |Y|$ with $y^{(j)}_{i_j} = 1$. However, the superposition in $|\psi_2\rangle$ is over elements of $K_{(|Y|)} = \{h \in H^\perp \mid h_{i_1} = \cdots = h_{i_{|Y|}} = 0\}$ and hence an application of $\mathcal{C}_Y$ ensures that we are not going to measure an element from Y again.

We note that by construction, $\mathbf{0} \in K_{(|Y|)}$. The $\mathbf{0}$ vector does not meet the requirements for the set Y because on all of its indices, it holds a 0. Hence, we cannot update Y with $\mathbf{0}$ and in that sense, measuring $\mathbf{0}$ does not give us any additional information that helps us to solve Simon's problem. We get around this by applying the quantum algorithm from Theorem 5. For each i with $0 \leq i < n$ we define $\chi_i : G \to \{0, 1\}$ with

$$\chi_i(x) = \begin{cases} 1, & x_i = 1 \\ 0, & x_i = 0 \end{cases}.$$

Based on the function χ_i, we construct a quantum operator $\mathcal{U}_{\chi_i}$ with $\mathcal{U}_{\chi_i} |x\rangle |q\rangle = |x\rangle |q \oplus \chi_i(x)\rangle$. This operator can easily be translated into a quantum circuit with size one, we simply apply the *CNOT* gate with the control qubit being $|x_i\rangle$ and the target qubit being $|q\rangle$. Moreover, we wrap the quantum algorithm we constructed so far into a fresh operator $\mathcal{A}$ with $\mathcal{A} = \mathcal{C}_Y \mathcal{B}$ and thus $\mathcal{A} |\psi_0\rangle = |\psi_2\rangle$.

We recall that the superposition in $|\psi_2\rangle$ is over elements of $K_{(|Y|)}$ and fix an arbitrary i with $1 \leq i < n$. Let $A = \{h \in K_{(|Y|)} \mid h_i = 1\}$ and $B = \{h \in K_{(|Y|)} \mid h_i = 0\}$. We analyze the superposition in $|\psi_2\rangle$ and distinguish between two cases.

1. The superposition is over states from B only.

2. The superposition is both over states from A and B. This follows from $K_{(|Y|)}$ being a group and the Equal Cardinality Lemma (Lemma 11).

Note that by the Equal Cardinality Lemma, the superposition in $|\psi_2\rangle$ can not be over states from A only. By construction, all states from A can be used to update the set Y. Additionally, those states in B can be used for the update which are different from the zero vector. We would like a guarantee that we measure a usable state, which we currently do not have, because in both cases one and two, the superposition of the quantum computer contains states from B. We are now going to use Theorem 5 to circumvent this.

Let $\mathcal{D}_i$ be the quantum algorithm we get by instantiating Theorem 5 with $\mathcal{A}$ and $\mathcal{U}_{\chi_i}$. By Theorem 5, if we apply $\mathcal{D}_i$ to $|\psi_0\rangle$, end up in the state $(2i(1-a)-1)\,|A\rangle + i(1-2a)\,|B\rangle$, where a, b are the probabilities to measure a state from A or B respectively. We apply this to the two cases described above.

1. If states from A do not exist, then $a = 0$ and $\mathcal{D}_i\,|\psi_0\rangle$ puts the quantum register in a superposition state of the form

$$(2i(1-0)-1)\,|A\rangle + i(1-2\cdot 0)\,|B\rangle = i\,|B\rangle\,.$$

2. If there are as many states in A as in B, then we have $a = b = \frac{1}{2}$. Hence when we apply $\mathcal{D}_i\,|\psi_0\rangle$, the quantum computer is in a superposition state of the form

$$(2i(1-\frac{1}{2})-1)\,|A\rangle + i(1-2\cdot\frac{1}{2})\,|B\rangle = (i-1)\,|A\rangle\,.$$

This means, if there exists some element $y^{(|Y|+1)}$ which is not yet contained in Y and which satisfies $y_i^{(|Y|+1)} = 1$, we are guaranteed to measure it after one application of $\mathcal{D}_i$. With this observation in place, we can construct the witness for the algorithm from the statement of this theorem.

Algorithm 8.1: GET_NEW_BASIS_ELEMENT, witness for Theorem 7

Input : Y a (not necessarily strict) subset from a basis of $H^\perp$.
Input : $\mathcal{U}_\rho$ a blackbox quantum implementation of ρ.
Output : An element from the basis of $H^\perp$ which is not in Y. If such an element does not exist $\mathbf{0}$.

```
for i = 0; i++; i < n do
    D_i ⟵ CONSTRUCT_QUANTUM_CIRCUIT(i, Y, U_ρ)
    y ⟵ RUN_QUANTUM_CIRCUIT_AND_MEASURE_FIRST_REGISTER(D_i)
    if y_i ≠ 0 then
        return y
return 0
```

In Algorithm 8.1, the CONSTRUCT_QUANTUM_CIRCUIT subroutine constructs the quantum circuit for the quantum operator $\mathcal{D}_i$. The subroutine RUN_QUANTUM_CIRCUIT_AND_MEASURE_FIRST_REGISTER then runs that circuit on a quantum computer with initial state $|\psi_0\rangle = |0\rangle^{\otimes n} |0\rangle^{\otimes m} |0\rangle^{\otimes |Y|}$ and returns the measurement result from the first register.

Suppose Y does not generate $H^\perp$. Then there exists some element $y \in H^\perp$ with $y \notin \langle Y \rangle$. We know that $y \neq \mathbf{0}$ because $\mathbf{0} \in \langle Y \rangle$. Hence, there must exist some index i such that $y_i = 1$. Running $\mathcal{D}_i |\psi_0\rangle$ results in a superposition of elements with a value 1 at index i in the first quantum register. By the K-Y-H-Lemma (Lemma 18), we know that all of those elements are in $H^\perp \setminus \langle Y \rangle$. Measuring the first register will yield a random such element and Algorithm 8.1 then returns it.

Suppose Y generates $H^\perp$. Then there exists no non-zero element $y \in H^\perp$ with $y \notin \langle Y \rangle$. Hence, for all i with $0 \leq i < n$, running $\mathcal{D}_i |\psi_0\rangle$ produces a superposition of elements with 0 at index i. Algorithm 8.1 in that case returns the zero vector.

We still need to prove the circuit depth claims for Algorithm 8.1. Fix an arbitrary i with $0 \leq i < n$ and consider the circuit size of $\mathcal{D}_i$. By construction of $\mathcal{D}_i$ and by Theorem 6, the depth of the circuit of $\mathcal{D}_i$ is linear in n, constant in the depth of $\mathcal{A}$ and constant in the depth of $\mathcal{U}_{\chi_i}$. We first analyze the circuit depth of $\mathcal{A}$ and $\mathcal{A}^{-1}$.

The first step of $\mathcal{A}$, where we perform the state change from $|\psi_0\rangle$ to $|\psi_1\rangle$, corresponds to Algorithm 5.1. By Theorem 2, the circuit depth of that algorithm is the depth of $\mathcal{U}_\rho$ plus a constant.

In the second step of $\mathcal{A}$, where we go from $|\psi_1\rangle$ to $|\psi_2\rangle$, we apply $\mathcal{C}_j$ for each $j \leq |Y|$. The operator $\mathcal{C}_j$ is defined as the quantum algorithm described in Corollary 4 of the Blocking Clause Lemma (Lemma 12), where the singled-out non-zero element is $y^{(j)}$. By Lemma 12 we get that the circuit depth of $\mathcal{C}_j$ is linear in n for each $j \leq |Y|$. By construction $|Y| \leq n$ and hence we need to apply $\mathcal{C}_j$ at most n times. Consequently, going from $|\psi_1\rangle$ to $|\psi_2\rangle$ takes a number of quantum computation steps quadratic in n.

Combining the results for the first and second step of $\mathcal{A}$, we get that one application of $\mathcal{A}$ in total takes one call to $\mathcal{U}_\rho$ plus a number of other quantum computation steps within $O(n^2 + C) = O(n^2)$ where C is some constant. Furthermore, as shown above, the operator $\mathcal{U}_{\chi_i}$ can be implemented using a single quantum gate. By Theorem 6, running $\mathcal{D}_i$ therefore takes only a constant number of calls to U_ρ and a number of other quantum computer steps (asymptotically) quadratic in n.

In Algorithm 8.1, we invoke $\mathcal{D}_i$ for all i with $0 \leq i < n$ in the worst case. Hence, in the worst case, we need n calls to U_ρ, plus a number of other quantum computation steps cubic in n. □

We note that GET_NEW_BASIS_ELEMENT (Algorithm 8.1) is already a hybrid quantum-classical algorithm. In each iteration of the for-loop, we first construct a quantum circuit on classical hardware, next we run that circuit on quantum hardware, and last we use classical hardware again in order to decide whether we continue the loop or not. Algorithm 8.1 also showcases a key difference between the original version of Simon's algorithm and the extended version: In the extended version, we generate a different quantum circuit in every loop iteration, whereas in the original version we run the same quantum circuit multiple times. So far we have a procedure that generates one additional element for our basis of $H^\perp$. We still need to wrap this procedure into a larger algorithm that constructs an entire basis of $H^\perp$. This is achieved by the following Algorithm 8.2, which was originally stated by Brassard and Høyer in Theorem 5 [13].

Algorithm 8.2: Quantum Part of the Deterministic Solution to Simon's Problem

Input : $\mathcal{U}_\rho$ a blackbox quantum implementation of ρ.
Output : Y a basis of the hidden subgoup of ρ.

$Y \longleftarrow \emptyset$
while $(y \longleftarrow$ *GET_NEW_BASIS_ELEMENT*$(Y, \mathcal{U}_\rho)) \neq \mathbf{0}$ **do**
  $Y \longleftarrow Y \cup \{y\}$
return Y

By Theorem 7, with every loop iteration, we update Y with a fresh basis element from $H^\perp$ that is linearly independent from all other vectors in Y. By the K-Y-H Lemma (Lemma 18), if GET_NEW_BASIS_ELEMENT returns the zero vector, then we have $\langle Y \rangle = H^\perp$.

As noted by Brassard and Høyer in [13], Algorithm 8.2 can be optimized. Following the pseudocode in Algorithm 8.1, on each invocation of GET_NEW_BASIS_ELEMENT we run $\mathcal{D}_i$ for each i with $0 \leq i < n$. However, we actually never have to execute $\mathcal{D}_i$ twice for the same i [13]. This is because of three reasons. First, if one application of $\mathcal{D}_i$ yields a bitstring with value 1 at index i, then because of Theorem 4, it does not make sense to run $\mathcal{D}_i$ again in future iterations, as all bitstrings with value 1 at index i will be eliminated from future superpositions. Second, if one application of $\mathcal{D}_i$ yields a nonzero bitstring that has a value 0 at index i, that bitstring can still be used to update the set

Y. Third, if one application of $\mathcal{D}_i$ returns the zero vector, then we know for sure that there exists no bitstring we have not measured so far that has a value 1 at index i.

8.3 Classical Post-Processing for the Extended Version of Simon's Problem

After a run of Algorithm 8.2, we obtain Y, a basis of $H^\perp$. We still need to convert this to a basis of H. In principle, this works completely analogous to the classical post-processing for the standard version of Simon's algorithm described in Section 4.2. However, since now the hidden subgroup can be of any size, the mathematical analysis becomes more complicated. We first present a lemma describing a way to transform $H^\perp$ into H.

Lemma 19. *Let H be any subgroup of the group $G = \langle \{0,1\}^n, \oplus \rangle$ and let $H^\perp$ be its orthogonal set. Then $(H^\perp)^\perp = H$.*

Proof. First we prove $H \subseteq (H^\perp)^\perp$. By definition of $(H^\perp)^\perp$, for any $x \in G$ we have

$$x \in (H^\perp)^\perp \iff x \odot g = 0 \quad \forall g \in H^\perp.$$

If $x \in H$, then for arbitrary $g \in H^\perp$, we have $x \odot g = 0$ and hence $x \in (H^\perp)^\perp$.

Next we prove that $(H^\perp)^\perp$ cannot contain more elements than those in H by analyzing the size of $(H^\perp)^\perp$. By Lemma 10, we have $|H^\perp| = \frac{|G|}{|H|}$. Moreover, by Lemma 17, $(H^\perp)^\perp$ is a group and hence we can apply Lemma 10 again to conclude

$$|(H^\perp)^\perp| = \frac{|G|}{|H^\perp|} = \frac{|G|}{\frac{|G|}{|H|}} = \frac{|G| \cdot |H|}{|G|} = |H|.$$

$(H^\perp)^\perp$ contains all elements of H, and it does not contain any others, hence $(H^\perp)^\perp = H$. □

We use the bitstrings from Y to construct a system of linear equations analogous to Section 4.2. By Lemma 19, the solution space to that system of equations is the hidden subgroup H. Note that in the case of the standard version of Simon's algorithm, after solving the system of linear equations, we had to run some implementation of ρ again in order to distinguish between the cases where ρ is bijective or not. This step becomes obsolete in the extended version of Simon's algorithm, since we are now guaranteed that Y is a complete basis of $H^\perp$ and the solution space to the system of equations corresponds precisely to H.

8.4 An Example Run of the Extended Version of Simon's Algorithm

We revisit the instance of Simon's problem from Section 4.3, but we first rephrase it as an instance of the extended version of Simon's problem. The oracle operator from

Section 4.3, $\mathcal{U}_\rho$, has the hidden subgroup $H = \{000, 001\}$. We calculate the cosets of H in $G = \{0,1\}^3$ to be $\{000, 001\}, \{010, 011\}, \{100, 101\}$ and $\{110, 111\}$, and we observe that $\mathcal{U}_\rho$ indeed behaves constant for each element of each coset and distinct for each coset. Additionally, we choose a set T of coset representatives as $T = \{000, 010, 100, 110\}$, and we calculate $H^\perp = \{000, 010, 100, 110\}$.

For Algorithm 8.2, we need three quantum registers. The first one is of size 3, holding input values of the oracle operator. The second one is of size 2 and holds output values of the oracle. Finally, we need another quantum register of dynamic size, depending on how many bitstrings we already measured.

We now run Algorithm 8.2. Entering the loop, we first execute GET_NEW_BASIS_ELEMENT($\emptyset$). In that function, we first run the quantum circuit for $\mathcal{D}_0$. By construction, $\mathcal{D}_0$ wraps the algorithm $\mathcal{C}_\emptyset\mathcal{B}$ from Theorem 7 in the algorithm $\mathcal{Q}$ from Theorem 5. Before we start with the calculations, we briefly recapitulate the quantum operators in use.

Algorithm $\mathcal{B}$ is the original version of Simon's algorithm, which puts the first register into a superposition of elements of $H^\perp$. More formally, by Theorem 3, in our setting we have

$$\mathcal{B}\left|000\right\rangle\left|00\right\rangle = \frac{1}{\sqrt{|T|}}\Big(\sum_{t\in T}\left|\phi_t H^\perp\right\rangle\left|\rho(t)\right\rangle\Big).$$

With algorithm $\mathcal{C}_\emptyset$, we remove already measured elements from $H^\perp$. In our case, since this is our first time executing GET_NEW_BASIS_ELEMENT, we do not yet have any bitstring to block. More formally, by Theorem 4, we have $\mathcal{C}_\emptyset = \mathcal{I}$ and the third quantum register is empty.

We first analyze the behavior of $\mathcal{B}$, and then we investigate the effects of wrapping $\mathcal{B}$ in $\mathcal{Q}_0$. For the initial state $\left|\psi_0\right\rangle = \left|000\right\rangle\left|00\right\rangle$, by Theorem 3, the operator $\mathcal{B}$ performs the state change

$$\begin{aligned}
\left|\psi_1\right\rangle = \mathcal{B}\left|\psi_0\right\rangle &= \frac{1}{\sqrt{|T|}}\Big(\sum_{t\in T}\left|\phi_t H^\perp\right\rangle\left|\rho(t)\right\rangle\Big) \\
&= \frac{1}{\sqrt{4}}\Big(\sum_{t\in T}\Big(\frac{1}{\sqrt{|H^\perp|}}\sum_{g\in H^\perp}(-1)^{g\odot t}\left|g\right\rangle\Big)\left|\rho(t)\right\rangle\Big) \\
&= \frac{1}{\sqrt{4}}\Big(\sum_{t\in T}\frac{1}{\sqrt{4}}\Big(\left(-1\right)^{000\odot t}\left|000\right\rangle + \left(-1\right)^{010\odot t}\left|010\right\rangle + \left(-1\right)^{100\odot t}\left|100\right\rangle \\
&\qquad\qquad + \left(-1\right)^{110\odot t}\left|110\right\rangle\Big)\left|\rho(t)\right\rangle\Big).
\end{aligned}$$

We now recall from Section 4.3 that $\rho(000) = 00$, $\rho(010) = 01$, $\rho(100) = 10$ and

$\rho(110) = 11$ and rewrite $|\psi_1\rangle$ to

$$\begin{aligned}|\psi_1\rangle = \frac{1}{\sqrt{4}}\Bigg(&\frac{1}{\sqrt{4}}\Big(|000\rangle + |010\rangle + |100\rangle + |110\rangle\Big)|00\rangle \\ &+ \frac{1}{\sqrt{4}}\Big(|000\rangle - |010\rangle + |100\rangle - |110\rangle\Big)|01\rangle \\ &+ \frac{1}{\sqrt{4}}\Big(|000\rangle + |010\rangle - |100\rangle - |110\rangle\Big)|10\rangle \\ &+ \frac{1}{\sqrt{4}}\Big(|000\rangle - |010\rangle - |100\rangle + |110\rangle\Big)|11\rangle\Bigg).\end{aligned}$$

The state $|\psi_1\rangle$ can be partitioned into two sets of states. First, we group those states that hold a 1 at index 0 of the first quantum register (the least significant bit in Qiskit bit order) into set A. Second, we group those states that hold a 0 at index 0 of the first quantum register into set B. We immediately realize that all states in $|\psi_1\rangle$ are in set B and that $A = \emptyset$. As a consequence, the probability to measure a state from A, denoted a, is zero. We now recapitulate Theorem 5. When we wrap $\mathcal{B}$ into $\mathcal{Q}_0$, we get

$$\mathcal{Q}_0\,|000\rangle\,|0\rangle = ((2i(1-a)-1)\,|A\rangle + i(1-2a)\,|B\rangle) = i\cdot|B\rangle = i\cdot|\psi_1\rangle\,.$$

Consequently, when we wrap $\mathcal{B}$ into $\mathcal{Q}_0$ and run $\mathcal{Q}_0\,|\psi_0\rangle$, the result is state $|\psi_1'\rangle$ which differs from $|\psi_1\rangle$ only in one way: The global phase is shifted by the imaginary unit i. When we now measure the first register from $|\psi_1'\rangle$, we measure any element from $H^\perp$. For this example, let us assume that we are particularly unlucky and measure the zero vector.

The algorithm now enters the second loop iteration and, which consists of constructing and running $\mathcal{Q}_1$. The algorithm $\mathcal{Q}_1$ is again constructed by wrapping $\mathcal{B}$ into the algorithm from Theorem 5. We thus inspect the state $|\psi_1\rangle$ again. This time, we group all bitstrings into set A that hold a 1 at index 1 (the middle qubit) of the first quantum register. All states with a bit value 0 at the same index are grouped into set B. We realize that precisely half the states in $|\psi_1\rangle$ are in A and the other half is in B (and all have the same amplitude), so $a = \frac{1}{2}$. By Theorem 5, we thus have

$$\begin{aligned}\mathcal{Q}_1\,|000\rangle\,|00\rangle &= ((2i(1-a)-1)\,|A\rangle + i(1-2a)\,|B\rangle) \\ &= (2i(1-\frac{1}{2})-1)\,|A\rangle + i(1-2\cdot\frac{1}{2})\,|B\rangle = (i-1)\,|A\rangle\,.\end{aligned}$$

More precisely, we end up in the state

$$\begin{aligned}|\psi_1'\rangle = \frac{1}{\sqrt{4}}\Bigg(&\frac{1}{\sqrt{4}}\Big(i\cdot|010\rangle - |010\rangle + i\cdot|110\rangle - |110\rangle\Big)|00\rangle\\ &+\frac{1}{\sqrt{4}}\Big(-i\cdot|010\rangle + |010\rangle - i\cdot|110\rangle + |110\rangle\Big)|01\rangle\\ &+\frac{1}{\sqrt{4}}\Big(i\cdot|010\rangle - |010\rangle - i\cdot|110\rangle + |110\rangle\Big)|10\rangle\\ &+\frac{1}{\sqrt{4}}\Big(-i\cdot|010\rangle + |010\rangle + i\cdot|110\rangle - |110\rangle\Big)|11\rangle\Bigg),\end{aligned}$$

and hence when we measure the first register, we are guaranteed to obtain a nonzero bitstring. For this example, let us assume we measure the bitstring $y^{(1)} = 110$. This gets returned by the GET_NEW_BASIS_ELEMENT($\emptyset$) routine, and we set $Y = \emptyset \cup \{y^{(1)}\} = \{110\}$.

Next we run GET_NEW_BASIS_ELEMENT($\{110\}$). To simplify the analysis, we use the fact that we only have to try each $\mathcal{Q}_i$ once and that it is hence only left to run $\mathcal{Q}_2$. Analogously to above, we first investigate the effects of applying the quantum algorithm $\mathcal{C}_1(\mathcal{B}\cdot\mathcal{I})$ to $|\psi_0\rangle$, and then we investigate the effects of wrapping it into $\mathcal{Q}_2$. Note that this time we do have the bitstring $y^{(1)}$, which we want to remove from the superposition over $H^\perp$ created by $\mathcal{B}$ and hence we are now working on three quantum registers. We have $|\psi_0\rangle = |000\rangle|00\rangle|0\rangle$ and thus

$$\begin{aligned}|\psi_1\rangle = |000\rangle|00\rangle|0\rangle &= (\mathcal{B}\otimes\mathcal{I})|\psi_0\rangle\\ &= \frac{1}{\sqrt{4}}\Bigg(\frac{1}{\sqrt{4}}\Big(|000\rangle + |010\rangle + |100\rangle + |110\rangle\Big)|00\rangle\\ &\quad+\frac{1}{\sqrt{4}}\Big(|000\rangle - |010\rangle + |100\rangle - |110\rangle\Big)|01\rangle\\ &\quad+\frac{1}{\sqrt{4}}\Big(|000\rangle + |010\rangle - |100\rangle - |110\rangle\Big)|10\rangle\\ &\quad+\frac{1}{\sqrt{4}}\Big(|000\rangle - |010\rangle - |100\rangle + |110\rangle\Big)|11\rangle\Bigg)|0\rangle\,.\end{aligned}$$

For the already known bitstring $y^{(1)} = 110$, we set $i_j = 1$ and hence $K_{(1)} = \{h \in H^\perp \mid h_1 = 0\} = \{000, 100\}$. In Theorem 7 we defined

$$\mathcal{C}_1\frac{1}{\sqrt{|T|}}\Big(\sum_{t\in T}\left|\phi_t H^\perp\right\rangle|\rho(t)\rangle\Big)|0\rangle = \frac{1}{\sqrt{|T|}}\Big(\sum_{t\in T}\left|\phi_t K_{(1)}\right\rangle|\rho(t)\rangle\Big)\left|t\odot y^{(1)}\right\rangle.$$

When we apply the quantum algorithm $\mathcal{C}_1$ to $|\psi_1\rangle$, we shrink the superposition in $|\psi_1\rangle$ to

$$\begin{aligned}|\psi_2\rangle &= \mathcal{C}_1 |\psi_1\rangle \\ &= \frac{1}{\sqrt{4}}\left(\frac{1}{\sqrt{2}}\Big(|000\rangle + |100\rangle\Big)|00\rangle|0\rangle + \frac{1}{\sqrt{2}}\Big(|000\rangle + |100\rangle\Big)|01\rangle|1\rangle\right. \\ &\quad \left. + \frac{1}{\sqrt{2}}\Big(|000\rangle - |100\rangle\Big)|10\rangle|1\rangle + \frac{1}{\sqrt{2}}\Big(|000\rangle - |100\rangle\Big)|11\rangle|0\rangle\right).\end{aligned}$$

We partition the states of $|\psi_2\rangle$ into two sets A (all states that hold a bit value 1 at index 3) and B (all states that hold a bit value 0 at index 3). We notice that exactly half of the states in $|\psi_2\rangle$ is in A and the other half is in B, hence when we wrap $\mathcal{C}_1 \otimes (\mathcal{B} \cdot \mathcal{I})$ into $\mathcal{Q}_3$, we end up in the state

$$\begin{aligned}|\psi_2'\rangle &= \mathcal{Q}_3 |\psi_0\rangle = (i-1)|A\rangle \\ &= \frac{1}{\sqrt{4}}\left(\frac{1}{\sqrt{2}}\Big(i \cdot |100\rangle - |100\rangle\Big)|00\rangle|0\rangle + \frac{1}{\sqrt{2}}\Big(i \cdot |100\rangle - |100\rangle\Big)|01\rangle|1\rangle\right. \\ &\quad \left. + \frac{1}{\sqrt{2}}\Big(-i \cdot |100\rangle + |100\rangle\Big)|10\rangle|1\rangle + \frac{1}{\sqrt{2}}\Big(-i \cdot |100\rangle - |100\rangle\Big)|11\rangle|0\rangle\right).\end{aligned}$$

When we measure the first register, we are guaranteed to measure the bitstring $y^{(2)} = 100$. Since we have run $\mathcal{Q}_i$ for each i with $0 \leq 2$, we know that we are done constructing the basis of $H^\perp$ and Algorithm 8.2 terminates, returning $Y = \{y^{(1)}, y^{(2)}\} = \{110, 100\}$. We use Y to form a system of linear equations

$$\begin{aligned}1 \cdot x_2 + 1 \cdot x_1 + 0 \cdot x_0 &\equiv 0 \pmod 2 \\ 1 \cdot x_2 + 0 \cdot x_1 + 0 \cdot x_0 &\equiv 0 \pmod 2.\end{aligned}$$

As in the original example, the solution space for this system is $\{000, 001\}$ and a generating set of this solution space is $\{001\}$, which is the solution to this instance of the extended version of Simon's problem.

8.5 Computational Complexity

The algorithm for the extended version of Simon's problem consists of two phases. First the quantum phase (Algorithm 8.2) and second the classical post-processing. We first analyze the circuit depth of Algorithm 8.2. Let $G = \{0,1\}^n$ and let $H \subseteq G$ be any hidden subgroup of G. In the worst case, $H = \{\mathbf{0}\}$ and thus by Lemma 10, $|H^\perp| = 2^n$. Any basis of $H^\perp$ then has size n, which means that we have to run GET_NEW_BASIS_ELEMENT exactly n times. By Theorem 7, one such execution of GET_NEW_BASIS_ELEMENT requires $O(n)$ calls of $\mathcal{U}_\rho$ and a number of other quantum computation steps within $O(n^3)$. Hence, executing GET_NEW_BASIS_ELEMENT n times costs $O(n^2)$ calls to $\mathcal{U}_\rho$ and a number of other quantum computation steps within $O(n^4)$. When we use the optimization from the end of Section 8.2, we only need $O(n)$ many calls to $\mathcal{U}_\rho$ and $O(n^3)$ other quantum computation steps.

The second phase of the algorithm is completely analogous to the classical post-processing for the standard version of Simon's algorithm. Obtaining a basis for the solution space of a system of linear equations can be done via Gaussian elimination in $O(n^3)$ computation steps [19]. Thus, we can place the extended version of Simon's problem in the complexity class $\mathcal{EQP}^{\mathcal{U}_\rho}$.

CHAPTER 9

Implementation

For this thesis, the algorithm for the extended version of Simon's problem consisting of Algorithm 8.2 and the procedure described in Section 8.3 were implemented in Python [24]. Generating the quantum circuits was done using Qiskit [21] and for the classical post-processing we used SymPy [25]. We already described almost all quantum circuits that are needed to run the algorithm:

- The circuit for the original version of Simon's algorithm is described in Section 5.1 in Algorithm 5.1.
- The circuit that implements a blocking clause to exclude already measured bitstrings from future measurements is described in Section 6.1 in the proof of Lemma 12.
- The circuit for the operator that shifts the phase of a register precisely if the register is in state $|\mathbf{0}\rangle$ (the quantum operator $\mathcal{S}_{\{0\}}$) is described in Section 7.2 in the proof of Lemma 16.
- The circuit for shifting the phase of a register precisely if the qubit with index i holds a $|1\rangle$ (the quantum operator $\mathcal{U}_{\chi_i}$) is described in Section 8.2 in the proof of Theorem 7.

There is one part missing. Up until now, we abstracted away the actual input for instances of Simon's problem, the oracle function implementations. This is no big deal for the original version of Simon's problem, since it is well studied, and possible oracle implementations can be found on most tutorial pages or in courses for quantum computing (e.g. the tutorial repository from Amazon Braket [2] or the IBM Quantum Learning platform [6]). The extended version of Simon's problem has received less attention and hence such out-of-the-box solutions do not already exist.

Hence, in this chapter we first describe a way to create oracle circuits for the extended version of Simon's problem. Next, we present a concrete example of the circuits being run for the extended version of Simon's algorithm. Last, we briefly discuss challenges for running those circuits on current NISQ hardware.

9.1 Oracle Implementation

In this section, we present a way to dynamically create valid oracles for the extended version of Simon's problem, given the hidden subgroup of the corresponding problem instance. Let ρ be a function that fulfills the promise from the extended version of Simon's problem. That is, we have $\rho : G \to R$ where G is the group $\langle \{0,1\}^n, \oplus \rangle$ and R is some set representable on a quantum computer. Furthermore, there exists a hidden subgroup H of G such that ρ behaves constant and distinct on each coset of H in G.

The first question we must address when designing a circuit for an oracle is how many qubits we need. The function ρ has domain G, that is, bitstrings of length n. Hence, to hold the input values of ρ, we need a quantum register of size n. The range of ρ is the set R, and since this is any arbitrary set representable on a quantum computer, we have a certain degree of freedom with respect to how many qubits we need to hold the output values of ρ. There is no upper bound for that register size, but we can give a lower bound. By Lemma 9, there are precisely $m = \frac{|G|}{|H|}$ cosets of H in G, and for each coset we need one unique output value for ρ. We assign each coset of H in G a number within the closed interval $[0, m-1]$, and we can model those m numbers on a quantum register in binary notation using a total of $log_2(m)$ qubits. Note that both $|G|$ and $|H|$ are powers of two, hence m is a power of two as well and $log_2(m)$ is a natural number. As a special case, we set $m = 1$ if $|H| = |G|$, because $log_2(1) = 0$ and we would like at least one output qubit. Technically, we also need a register that holds ancillary qubits for various computations in the oracle circuit. In order to keep the presentation concise, we ignore that ancillary register in the following analysis.

The second question of interest is how we design a circuit that maps inputs of ρ to their correct outputs. We first present a more abstract corresponding quantum algorithm and next we describe how to implement that algorithm as a quantum circuit. As a building block for the quantum algorithm, we define a quantum operator that works on two quantum registers of size n and m respectively. Intuitively, the first quantum register holds the input value for function ρ and the second quantum register holds its output value. For each coset $T^{(i)}$ of H in G where $0 \leq i < m$ and for each $t^{(j)} \in T^{(i)}$ where $0 \leq j < |T^{(i)}|$, define $\mathcal{T}_{i,j}$ as follows

$$\mathcal{T}_{i,j} |x\rangle |z\rangle = \begin{cases} |x\rangle |\text{bin}(i) \oplus z\rangle & \text{if } x = t^{(j)} \\ |x\rangle |z\rangle & \text{otherwise} \end{cases},$$

where $\text{bin}(i)$ denotes the binary representation of the natural number i. We further define $\mathcal{T}_i = (\mathcal{T}_{i,|T^{(i)}|-1} \cdots \mathcal{T}_{i,0})$ and $\mathcal{U}_\rho = \mathcal{T}_{m-1} \cdots \mathcal{T}_0$. We now prove that the quantum algorithm $\mathcal{U}_\rho$ is indeed a valid implementation of ρ.

Lemma 20. *The quantum algorithm $\mathcal{U}_\rho$ is a valid instance of the extended version of Simon's problem.*

Proof. Let $|x\rangle, |y\rangle$ be arbitrary states from the computational standard basis $\mathfrak{B}_n$. Furthermore, let $|z\rangle$ be an arbitrary state from $\mathfrak{B}_m$. Apply the operator $\mathcal{U}_\rho$ to $|x\rangle |z\rangle$ and $|y\rangle |z\rangle$ respectively and let the resulting states be $|x\rangle |z_x\rangle$ and $|y\rangle |z_y\rangle$ (note that by construction, $\mathcal{U}_\rho$ never changes the value held in the first quantum register). What we need to show is that $|z_x\rangle = |z_y\rangle$ precisely if both x and y are in the same coset of H in G.

Recall that by the Coset Lemma (Lemma 9), the cosets of H in G partition G and thus also $\mathfrak{B}_n$. Hence, there must exist cosets $T^{(i)}$ and $T^{(k)}$ such that $x \in T^{(i)}$ and $y \in T^{(k)}$. Note that any arbitrary operator $\mathcal{T}_\ell$, by construction, acts on input states from coset $T^{(\ell)}$ only. As a consequence, since we have $\mathcal{U}_\rho = \mathcal{T}_{m-1} \cdots \mathcal{T}_k \cdots \mathcal{T}_i \cdots \mathcal{T}_0$, when we examine the effect of $\mathcal{U}_\rho |x\rangle |z\rangle$, it is enough to just investigate the effects of $\mathcal{T}_i$ on $|x\rangle |z\rangle$. Analogously, when we examine the effects of $\mathcal{U}_\rho |y\rangle |z\rangle$, it is enough to investigate the effects of $\mathcal{T}_k |y\rangle |z\rangle$.

Assume that both x and y are in the same coset of H in G or more formally, $T^{(i)} = T^{(k)}$. Then we can write $x = t^{(j)}$ and $y = t^{(\ell)}$ for some $t^{(j)}, t^{(\ell)} \in T^{(i)}$. Thus, $\mathcal{T}_i$ has the form $\mathcal{T}_i = \mathcal{T}_{i,|T^{(i)}|-1} \cdots \mathcal{T}_{i,\ell} \cdots \mathcal{T}_{i,j} \cdots \mathcal{T}_{i,0}$. If the first register holds $|x\rangle$, then $\mathcal{T}_{i,j}$ is the only operator that has an effect in $\mathcal{T}_i$. Correspondingly, if the first register holds $|y\rangle$, then $\mathcal{T}_{i,\ell}$ is the only operator that has an effect in $\mathcal{T}_i$. Now we simply derive $\mathcal{T}_{i,j} |x\rangle |z\rangle = |x\rangle |\text{bin}(i) \oplus z\rangle$ and thus $|z_x\rangle = |\text{bin}(i) \oplus z\rangle$. Analogously, $\mathcal{T}_{i,\ell} |y\rangle |z\rangle = |y\rangle |\text{bin}(i) \oplus z\rangle$ and thus $|z_y\rangle = |\text{bin}(i) \oplus z\rangle$, and we have $|z_x\rangle = |z_y\rangle$.

Assume that x and y are not in the same coset of H in G or more formally, $T^{(i)} \neq T^{(k)}$. In the same way as above, we deduce $\mathcal{U}_\rho |x\rangle |z\rangle = \mathcal{T}_i |x\rangle |z\rangle = \mathcal{T}_{i,j} |x\rangle |z\rangle = |x\rangle |\text{bin}(i) \oplus z\rangle$ where we get j by rewriting x as some element $t^{(j)} \in T^{(i)}$. We have $|z_x\rangle = |\text{bin}(i) \oplus z\rangle$. Similarly, we get $\mathcal{U}_\rho |y\rangle |z\rangle = \mathcal{T}_k |y\rangle |z\rangle = \mathcal{T}_{k,\ell} |y\rangle |z\rangle = |y\rangle |\text{bin}(k) \oplus z\rangle$ where we get ℓ by rewriting x as some element $t^{(\ell)} \in T^{(k)}$. We have $|z_y\rangle = |\text{bin}(k) \oplus z\rangle$. From $T^{(i)} \neq T^{(k)}$ we get $i \neq k$ and thus also $\text{bin}(i) \neq \text{bin}(k)$ and hence $|z_x\rangle \neq |z_y\rangle$. □

Next, we describe how to translate $\mathcal{U}_\rho$ into a quantum circuit. Since the operator $\mathcal{U}_\rho$ is composed entirely of the operators $\mathcal{T}_{i,j}$ for cosets $T^{(i)}$ and elements $t^{(j)} \in T^{(i)}$ it is enough to present a circuit that implements $\mathcal{T}_{i,j}$. For each $t^{(j)}$ in $T^{(i)}$, we construct the following circuit, where $|x\rangle$ and $|z\rangle$ are the input and output registers respectively.

1. For each index k with $t_k^{(j)} = 0$ we add an X gate at index k of the input register.
2. For each index k where $\text{bin}(i) = 1$ we add an $X_{(x_{n-1} \wedge \cdots \wedge x_0), z_k}$ gate.
3. Reset the input qubits by repeating step 1.

Suppose $|x\rangle$ holds the value $t^{(j)}$. Then after step 1., all qubits of $|x\rangle$ are in state $|1\rangle$. Consequently, all controlled X gates which are added in step 2., get activated and by

construction of step 2., the output register holds the value $|\text{bin}(i) \oplus z\rangle$. Since $\mathcal{X}\mathcal{X} = \mathcal{I}$, after step 3., all qubits in the input register are reset to their original values.

Suppose $|x\rangle$ does not hold the value $t^{(j)}$. In this case, the controlled X gates introduced in step 2. do not get activated. To see this, note that since $x \neq t^{(j)}$, there must exist at least one index k such that $x_k \neq t_k^{(j)}$. If $x_k = 0$, then $t_k^{(j)} = 1$ and hence in step 1., no X gate was added at index k of the input register. Since $|x_k\rangle = |0\rangle$, and since the controlled X gates are controlled by all qubits in the input register, step 2. has no effect. If $x_k = 1$, then $t_k^{(j)} = 0$. By construction of step 1., the circuit has an X gate at index k and hence after step 1., we have $|x_k\rangle = |0\rangle$. Analogous to before, the controlled-X gates in step 2. do not get activated. As explained above, step 3. resets the input register to its original state again.

Note that step 2. relies on multi-controlled X gates, which are non-standard, meaning that we have to simulate them. We use the same simulation method as described in the proof of Lemma 16. The circuit for $\mathcal{T}_{i,j}$ can be optimized in one aspect: Since $\mathbf{0} \oplus x = x$ for arbitrary $x \in G$, we do not have to create any circuit for $\mathcal{T}_0$.

Figure 9.1 depicts an example circuit for an oracle implementing the hidden subgroup $H = \{000, 001, 010, 011\}$. The cosets of H in G are $T^{(0)} = H$ and $T^{(1)} = \{100, 101, 110, 111\}$. In the circuit, the first register is the input register $|x\rangle$. Since we are operating in group $G = \langle\{0,1\}^3, \oplus\rangle$, it has size $n = 3$. The second register $|z\rangle$ is the output register. The hidden subgroup H has two different cosets in G, hence $m = 2$ and we can model the natural numbers 0 and 1 using just a single qubit (note that $\text{bin}(0) = 0$ and $\text{bin}(1) = 1$). Therefore, $|z\rangle$ has size 1. The last register of size 1 is an ancillary register.

Note that if the input register in Figure 9.1 holds a value from $T^{(0)}$, the depicted circuit does nothing and after the circuit was executed, the output qubit will hold the value $|z_0 \oplus \text{bin}(0)\rangle = |z_0 \oplus 0\rangle = |z\rangle$. Otherwise, if the input register holds a value from $T^{(1)}$, the output qubit will hold the value $|z_0 \oplus \text{bin}(1)\rangle = |z_0 \oplus 1\rangle$.

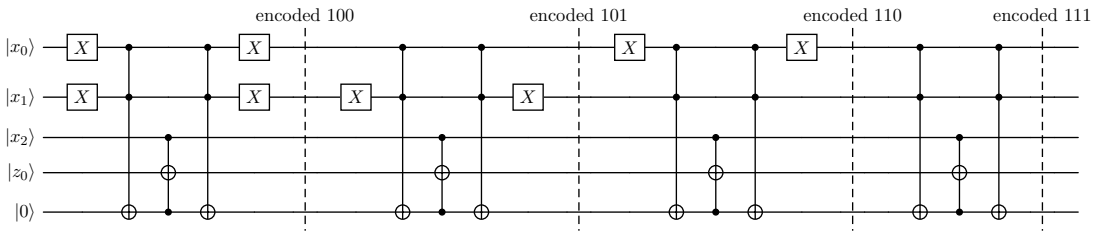


Figure 9.1: Circuit for $\mathcal{U}_\rho$ where $H = \{000, 001, 010, 100\}$ with the cosets $T^{(0)} = H$ and $T^{(1)} = \{100, 101, 110, 111\}$, and $\rho(x) = 0$ for $x \in T^{(0)}$ and $\rho(x) = 1$ for $x \in T^{(1)}$. Least significant qubit is topmost.

9.2 Circuit Examples

We present another example of the extended version of Simon's algorithm. This time, different to Section 4.3 and Section 8.4, we do not focus on the mathematical analysis of the algorithm, but rather on the generated quantum circuits. As at the end of Section 9.1, let $\rho : G = \{0,1\}^3 \to \{0,1\}$ with $H = \{000, 001, 010, 011\}$, $H^\perp = \{000, 100\}$ and $\rho(000) = \rho(001) = \rho(010) = \rho(011) = 0$ and $\rho(100) = \rho(101) = \rho(110) = \rho(111) = 1$.

We run Algorithm 8.2 with the optimization that we need to run $\mathcal{D}_i$ only once for each i with $0 \leq i < 3$. First we execute GET_NEW_BASIS_ELEMENT($\emptyset$) and thus execute the quantum algorithm $\mathcal{D}_0$. The quantum circuit for $\mathcal{D}_0$ is depicted in Figure 9.2 (A), where the circuit for $\mathcal{U}_\rho$ is the same as presented in Figure 9.1.

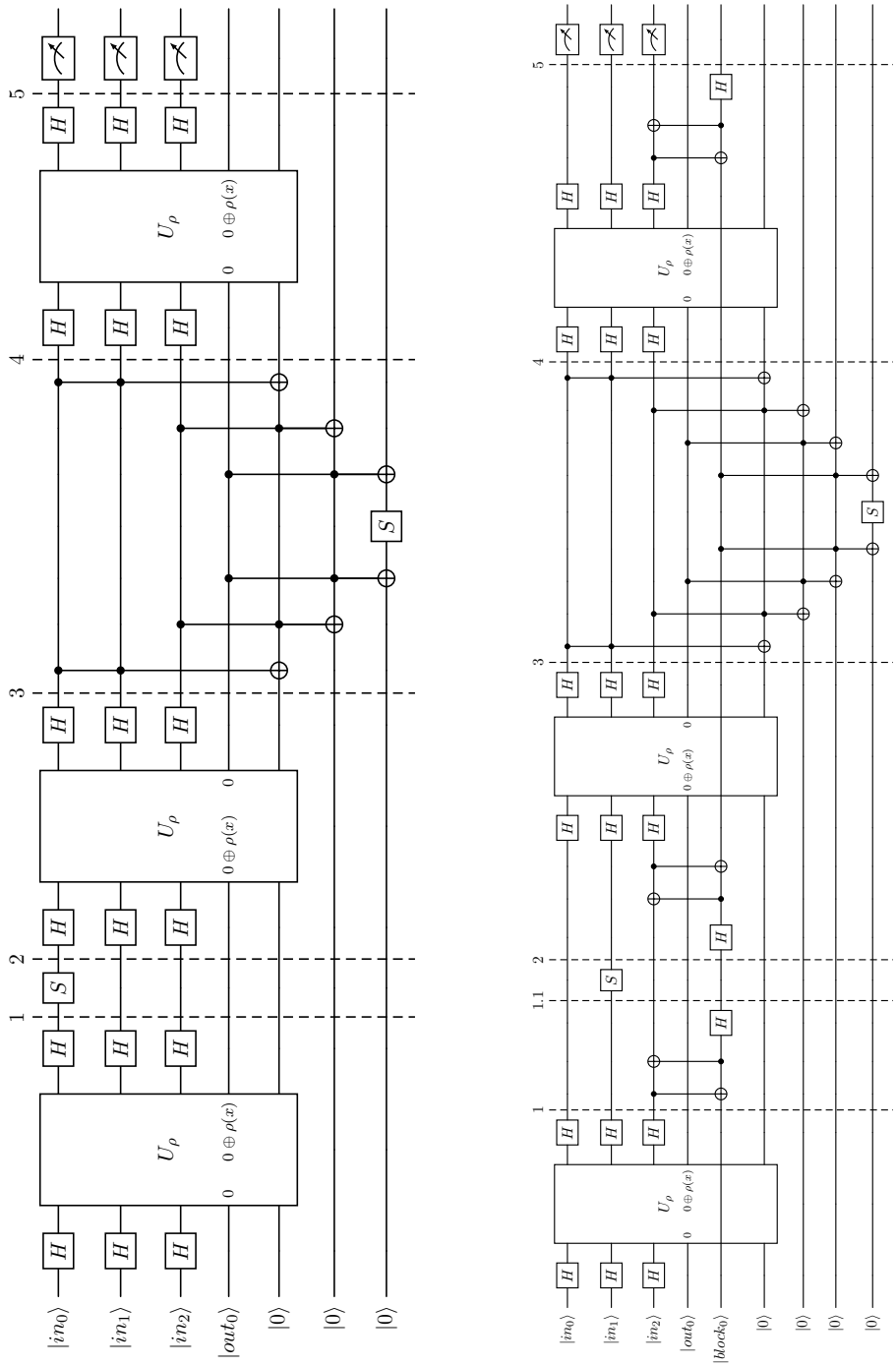
The quantum gates before barrier 1. implement the standard version of Simon's algorithm (without the measurements at the end). In the very first iteration of the algorithm, there are no blocking clauses yet, hence this step corresponds to the quantum algorithm $\mathcal{A}$ from the proof of Theorem 7. The single S gate between barriers 1. and 2. serves as the implementation of the quantum operator $\mathcal{U}_{\chi_0}$ described in the proof of Theorem 7. The quantum gates between barriers 2. and 3. correspond to $\mathcal{A}^{-1}$. The quantum gates between barriers 3. and 4. implement $\mathcal{S}_{\{0\}}$, where we shift the global phase by i precisely if all qubits in all working registers (that is the input register and the output register) hold the value $|0\rangle$. The quantum gates between barriers 4. and 5. correspond to one application of $\mathcal{A}$ again. The entire circuit thus implements the quantum algorithm $D_0 = \mathcal{A}\mathcal{S}_{\{0\}}\mathcal{A}^{-1}\mathcal{U}_{\chi_0}\mathcal{A}$.

After the application of $\mathcal{D}_0$, the state of the quantum computer is a superposition over elements from $H^\perp = \{000, 100\}$. Assume that we get lucky and measure the bitstring $y^{(1)} = 100$. Since $y^{(1)}$ holds a 1 at index 2 (the most significant qubit), all that is left to do is to run $\mathcal{D}_1$ in the context of GET_NEW_BASIS_ELEMENT($\{100\}$). The corresponding circuit is shown in Figure 9.2 (B). We immediately note how the circuit changes: First, there is one additional qubit which we need for the implementation of the blocking clause for $y^{(1)}$. That blocking clause is implemented with the gates between barriers 1. and 1.1. Second, the S gate between barriers 1.1 and 2. is now at index 1, since we are running a circuit for $\mathcal{D}_1$. Third, since $\mathcal{S}_{\{0\}}$ is now operating on the input register, the output register and additionally on the blocking clause register, we need one additional ancillary qubit for its implementation.

By Theorem 7, we are guaranteed to measure the zero vector after one run of $\mathcal{D}_1$. Moreover, we do not have to run $\mathcal{D}_2$, since $y_2^{(1)} = 1$ and all bitstrings with a 1 at index 2 are blocked from future measurements. We thus proceed to construct a system of equations

$$1 \cdot x_2 + 0 \cdot x_1 + 0 \cdot x_0 \equiv 0 \pmod 2$$

with the solution set $\{000, 001, 010, 011\}$. Any subset spanning this set is the solution to this instance of the extended version of Simon's problem (e.g. $\{001, 010\}$).



(A) First circuit (B) Second circuit

Figure 9.2: The quantum circuits generated for the extended version of Simon's problem where $H = \{000, 001, 010, 011\}$, least significant qubit is topmost

9.3 Considerations for NISQ Hardware

The implementation for the extended version of Simon's algorithm [24] was developed and tested using the noise-free quantum computing simulator Qiskit Aer [21]. The local simulation setup consisted of an Intel i7-13700H with 64 GiB RAM using Python 3.10.12 on Ubuntu 22.04.5 LTS. We found local simulations to be feasible for hidden subgroups of $\langle \{0,1\}^n, \oplus \rangle$ with $2 \leq n \leq 5$. For larger n, simulating a quantum computer on classical hardware is very computationally expensive. A main reason for this is that slightly increasing n increases the number of qubits needed for the generated circuits substantially. A small hidden subgroup of a large group has many cosets (recall Lemma 9) and hence oracle implementations need many output qubits. Moreover, a small hidden subgroup of a large group has a large orthogonal group (recall Lemma 10) and hence in such a situation we need many blocking clauses. Note also that each increase of any working register automatically increases the number of ancillary qubits needed for the simulation of $\mathcal{S}_{\{0\}}$, hence for local simulations without a GPU we quickly approach the limits of what is possible.

Next, we tested the implementation against simulators of the current state-of-the-art quantum computers IBM Eagle r3 [4] and IonQ Aria [5]. Recall that the extended version of Simon's problem is in $\mathcal{EQP}^{\mathcal{U}_\rho}$ and thus (for efficient implementations of $\mathcal{U}_\rho$) tractable from a purely complexity-theoretic point of view. Consider now, with actual NISQ hardware limitations in mind, the concrete problem instance from Section 9.2. The circuit in Figure 9.2 (A) has width 7 and depth 69, and the circuit in Figure 9.2 (B) has width 9 and depth 79. IBM rates their Eagle QPU at QV 128 [4], so both circuits, on paper, should be executable there. Note that our circuit sizes are calculated *before* transpilation to any particular QPU, and that the QV score depends on un-transpiled circuits as well [16, 28]. IonQ uses a specialized benchmark value similar to quantum volume called *algorithmic qubits (AQ)* [1], and they rate their Aria QPU at AQ 25 [5]. Intuitively, this means that Aria should be able to successfully execute all circuits with width up to 25 and depth up to $25^2 = 625$. Hence, on paper, both circuits should be executable on IonQ Aria as well. For all the following results, the standard Qiskit transpiler was used to translate circuits to the instruction sets supported by the corresponding quantum computers.

Figure 9.3 shows the measurement results from running the circuit in Figure 9.2 (A) 1024 times. We would expect to only sample bitstrings from $H^\perp$, which is $\{000, 100\}$ in that particular scenario. As we can see, this is only the case with the noise-free Qiskit Aer simulator. The results for both the simulators of IBM Eagle and IonQ Aria showcase significant noise. No bitstring outside $H^\perp$ has probability zero, and the total probability to measure a noisy result is around 64% for IBM Eagle and around 44% for IonQ Aria.

We observe similar results in Figure 9.4, which depicts the measurement results from running the circuit in Figure 9.2 (B). We would expect to only measure the zero vector (since 100 already generates $H^\perp$ in this scenario and 100 is blocked), but this is only achieved with the noise-free Qiskit Aer simulator. None of the real-device simulators give

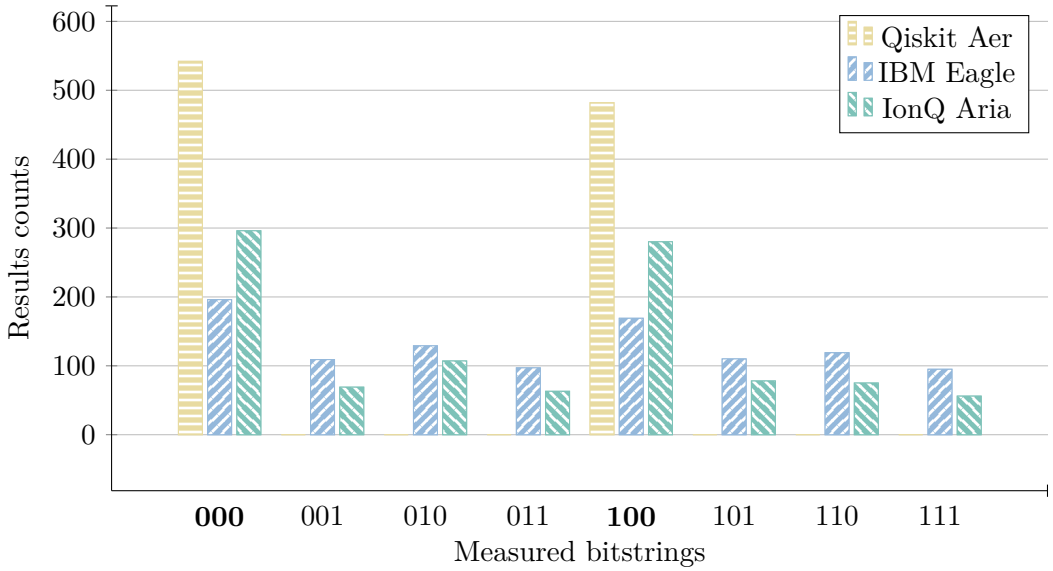


Figure 9.3: Measurement results from the quantum circuit in Figure 9.2 (A), all measurements obtained on simulators

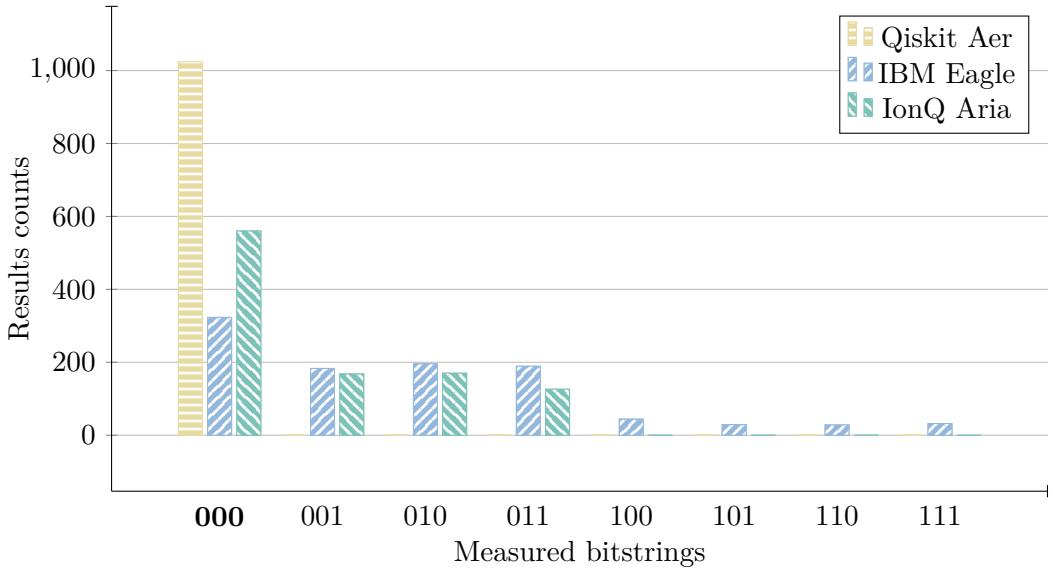


Figure 9.4: Measurement results from the quantum circuit in Figure 9.2 (B), all measurements obtained on simulators

us a guarantee to only measure 000. Furthermore, on the simulator for IBM Eagle, still each bitstring from $\{0,1\}^3$ is a possible measurement result and the total probability to measure a noisy result is around 68%. Interestingly, for the simulator for the IonQ Aria QPU, the bitstrings that are blocked by 100 (those with a value 1 at index 2, counting

from right to left), have now probability zero. However, we still do not get a guarantee to measure the zero vector itself, the probability for a noisy result is around 45%.

On paper (and on the noiseless Qiskit Aer simulator), the extended version of Simon's algorithm is deterministic. However, the measurement results from Figure 9.3 and 9.4 tell us that on real NISQ hardware, we still have to expect a high error probability. This issue is not unique to our setting, and a common mitigation stragegy is to not run a quantum circuit precisely once, but multiple times (as was done for collecting data for Figures 9.3 and 9.4). In that scenario, the result of running a quantum circuit is then often defined as the bitstring that was measured most often, which is also what our implementation does. Hence, despite the high error probabilities on both circuits depicted in Figure 9.2, our implementation picks a correct bitstring at each step and successfully solves this particular instance of Simon's problem.

Motivated by this fact, we also tested our implementation on a real IBM Eagle r3 QPU. The results of running the circuits in Figure 9.2 are depicted in Figure 9.5 and Figure 9.6, where we contrast the measurements obtained from the actual QPU with those from the Qiskit Aer simulator. The results are humbling, to say the least. In Figure 9.5 we observe that none of the desired bitstrings 000 and 100 are measured significantly more often than the rest. The opposite is the case, the bitstrings that are measured most often are noisy ones. We also cannot observe any clear pattern in the output, all possible measurement results seem to be roughly distributed uniformly. We also note that in this case, the actual IBM Eagle QPU performs significantly worse than we would expect from analyzing the corresponding simulator results, and now our implementation would choose a noisy bitstring, and thus reconstruct a wrong hidden subgroup.

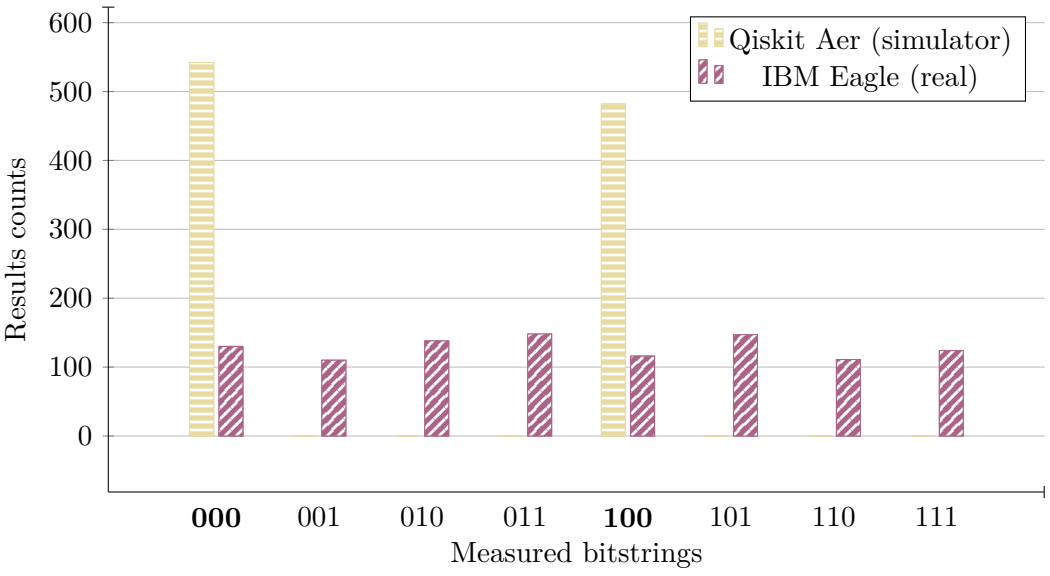


Figure 9.5: Measurement results from the quantum circuit in Figure 9.2 (A)

The results in Figure 9.6 are more in line with what we would expect from the simulations prior. We again have the problem that the only desired bitstring 000 is not measured most often, and thus our implementation would again choose a noisy bitstring and return a wrong result. However, a pattern we could observe in Figure 9.4 is also visible in Figure 9.6: The bitstrings blocked by the bitstring 100 are measured significantly less often than the rest.

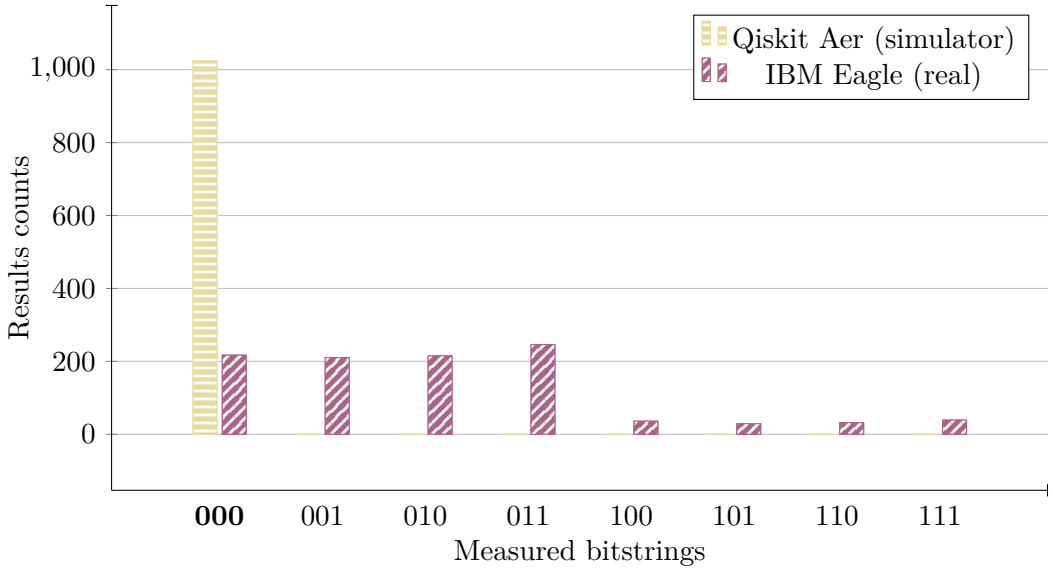


Figure 9.6: Measurement results from the quantum circuit in Figure 9.2 (B)

A detailed analysis of why the extended version of Simon's algorithm is such a challenge for NISQ hardware is outside the scope of this thesis, but we note the following. Our test circuits have width 7 and 9 only, which is significantly less than the 128 qubits available on IBM Eagle [4] or the 25 available on IonQ Aria [5], hence circuit width should not be the main obstacle. However, our circuits have depth 69 and 79 respectively, which appears to cause substantial noise due to decoherence effects and imprecise gates. Moreover, our circuits make heavy use of the three-qubit $CCNOT$ gate, which is non-standard both on IBM Eagle and IonQ Aria, and thus needs to be (expensively) simulated there.

CHAPTER 10

Conclusion

The standard version of Simon's problem [32] is well-studied and part of many introductory courses or tutorials on quantum computing. Hence, quite a lot of effort has gone into making Simon's algorithm accessible for a general public, and many well-documented implementations already exist [6, 2]. The same was not the case for the extended version of Simon's problem [13] until now. Although this extended version proved to be influential for further research in quantum computing, eventually leading to the discovery of the amplitude amplification technique [14, 18] and to the development of the hidden subgroup problem framework [27], so far no reference implementation for the extended version of Simon's algorithm was published. While the original version is usually covered in detail in textbooks on quantum computing (e.g. [20]), the extended version is only mentioned as a footnote, if at all.

This might be because of the first key insight from writing this thesis: the mathematical analysis of the extended version of Simon's algorithm is substantially more complex than the analysis of the original one. For the extended version, we require sophisticated results from linear algebra, from the structure of the algebraic group $\langle \{0,1\}^n, \oplus \rangle$ and its subgroups, and from quantum computing in general. The extended version of Simon's algorithm has many moving parts, which makes it particularly hard to engage with.

Back in the 1990s, when the different versions of Simon's algorithm were originally proposed, actual quantum hardware did not exist at all. There were also no corresponding toolchains for the development of quantum algorithms, and thus 'developing' meant giving an abstract mathematical description. Our second finding is that quantum toolchains now exist and that they can be pleasant to work with. Toolchains like IBM Qiskit [21] provide us with a straightforward way to create code runnable on actual quantum hardware, and the entire process is much closer to classical software development.

Our next finding is strongly related to the workflows of classical software development as well. A classical development paradigm is to first specify test cases and only then

start to work on any implementation (known as *test driven development* [9]). If we want to introduce the same process when implementing an algorithm to solve an oracle problem, we need some corresponding oracle implementations as test instances first. In quantum computing, for some well-studied quantum algorithms for oracle problems like the original version of Simon's algorithm [32] or the algorithm of Shor [31], concrete oracle implementations already exist [2, 6, 10]. However, for less known oracle problems like the extended version of Simon's problem, it is common practice to keep the oracle details abstract and to not actually care about their implementation. Hence, we needed to come up with our own test oracle implementation from scratch.

Lastly, we want to touch on our experimental results. On a noise-free quantum computer simulator our implementation works precisely as expected. That is, from each run of the quantum computer, we are guaranteed to measure a bitstring which gets us one step closer to solving an instance of the extended version of Simon's problem. In other words, on a noise-free simulator, our implementation of the extended version of Simon's algorithm is indeed deterministic. The results from the simulators of actual NISQ devices, while not ideal, also are in line with what we would expect. It is common knowledge that NISQ machines suffer severely from decoherence and gate infidelity [22], so it comes as no surprise that our implementation performs worse there. This could be in part because the quantum circuits generated by our implementation tend to be rather deep, and they rely heavily on the three-qubit $CCNOT$ gate, which needs to be expensively simulated on most machines. For both, we do not see an immediately obvious fix.

What comes as a disappointment are the results we got on the real IBM Eagle [4] QPU. Our results show that real NISQ hardware is not yet mature enough to run complex quantum algorithms like the extended version of Simon's algorithm successfully, not even for toy instances. Further research into better quantum hardware is needed, in order to make this technology applicable and attractive for real-world use cases.

Overview of Generative AI Tools Used

The bars of the charts in figures 9.3, 9.4, 9.5 and 9.6 contain patterns in order to make them better distinguishable on a black and white printout. ChatGPT [1] was used to find the correct Latex commands to create such patterns.

In the implementation [24] we make use of the static code analysis tool Pylint [2] in order to improve the overall code quality and in order to catch bugs early (note that Pylint is not a generative AI tool itself). As is common practice in software engineering, we exclude test code from the Pylint analysis, which is done via a configuration file called *.pylintrc*. ChatGPT was used to generate that configuration file.

Otherwise, no generative AI was used.

[1] `https://chatgpt.com/`, last accessed at 2025-04-17
[2] `https://www.pylint.org/`, last accessed at 2025-01-05

Bibliography

[1] Algorithmic Qubits. https://ionq.com/resources/algorithmic-qubits-a-better-single-number-metric. Accessed: 2025-03-27.

[2] Braket Tutorials Github. https://github.com/amazon-braket/amazon-braket-examples/. Accessed: 2025-03-22.

[3] Exploring Simon's Algorithm with Daniel Simon. https://aws.amazon.com/blogs/quantum-computing/simons-algorithm/. Accessed: 2024-09-05.

[4] IBM Eagle. https://docs.quantum.ibm.com/guides/processor-types#eagle. Accessed: 2025-04-03.

[5] IonQ Aria. https://ionq.com/quantum-systems/aria. Accessed: 2025-03-27.

[6] Quantum query algorithms. https://learning.quantum.ibm.com/course/fundamentals-of-quantum-algorithms/quantum-query-algorithms. Accessed: 2025-03-22.

[7] Sheldon Axler. *Linear Algebra Done Right*. Springer Nature, 2024. https://linear.axler.net/LADR4e.pdf.

[8] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457, 1995. https://arxiv.org/pdf/quant-ph/9503016.pdf.

[9] Kent Beck. *Test Driven Development: By Example*. The Addison-Wesley signature series. Addison-Wesley, Boston, 1st edition edition, 2003. https://learning.oreilly.com/library/view/test-driven-development/0321146530/.

[10] David Beckman, Amalavoyal N. Chari, Srikrishna Devabhaktuni, and John Preskill. Efficient networks for quantum factoring. *Physical Review A*, 54:1034–1063, Aug

1996. https://journals.aps.org/pra/abstract/10.1103/PhysRevA.54.1034.

[11] Ethan Bernstein and Umesh Vazirani. Quantum Complexity Theory. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of Computing*, pages 11–20, 1993. https://dl.acm.org/doi/pdf/10.1145/167088.167097.

[12] André Berthiaume and Gilles Brassard. Oracle Quantum Computing. *Journal of Modern Optics*, 41(12):2521–2535, 1994. https://doi.org/10.1080/09500349414552351.

[13] Gilles Brassard and Peter Høyer. An exact quantum polynomial-time algorithm for Simon's problem. In *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems*, pages 12–23. IEEE, 1997. https://arxiv.org/abs/quant-ph/9704027v1.

[14] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum Amplitude Amplification and Estimation, 2002. https://arxiv.org/pdf/quant-ph/0005055.

[15] André Chailloux, María Naya-Plasencia, and André Schrottenloher. An Efficient Quantum Collision Search Algorithm and Implications on Symmetric Cryptography. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 211–240, Cham, 2017. Springer International Publishing. https://link.springer.com/chapter/10.1007/978-3-319-70697-9_8.

[16] Andrew W. Cross, Lev S. Bishop, Sarah Sheldon, Paul D. Nation, and Jay M. Gambetta. Validating quantum computers using randomized model circuits. *Physical Review A*, 100:032328, Sep 2019. https://link.aps.org/doi/10.1103/PhysRevA.100.032328.

[17] David Deutsch. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985. https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1985.0070.

[18] Lov K. Grover. Quantum Computers Can Search Rapidly by Using Almost Any Transformation. *Physical Review Letters*, 80(19):4329, 1998. https://journals.aps.org/prl/pdf/10.1103/PhysRevLett.80.4329.

[19] Nicholas J. Higham. Gaussian elimination. *WIREs Computational Statistics*, 3(3):230–238, 2011. https://wires.onlinelibrary.wiley.com/doi/full/10.1002/wics.164.

[20] Matthias Homeister. *Quantum Computing verstehen*. Springer, 2008. https://link.springer.com/book/10.1007/978-3-658-36434-2.

[21] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J. Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D. Nation, Lev S. Bishop, Andrew W. Cross, Blake R. Johnson, and Jay M. Gambetta. Quantum computing with Qiskit, 2024. https://github.com/Qiskit/qiskit-aer.

[22] Frank Leymann and Johanna Barzen. The bitter truth about gate-based quantum algorithms in the NISQ era. *Quantum Science and Technology*, 5(4):044007, Sep 2020. https://dx.doi.org/10.1088/2058-9565/abae7d.

[23] Thomas Lubinski, Sonika Johri, Paul Varosy, Jeremiah Coleman, Luning Zhao, Jason Necaise, Charles H. Baldwin, Karl Mayer, and Timothy Proctor. Application-Oriented Performance Benchmarks for Quantum Computing. *IEEE Transactions on Quantum Engineering*, 4:1–32, 2023. https://ieeexplore.ieee.org/abstract/document/10061574.

[24] Oskar Mayer. Reference implementation for the extended version of simon's algorithm, April 2025. https://doi.org/10.5281/zenodo.15235735.

[25] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in Python. *PeerJ Computer Science*, 3:e103, January 2017. https://docs.sympy.org/latest/index.html.

[26] Takashi Mihara and Shao Chin Sung. Deterministic polynomial-time quantum algorithms for Simon's problem. *Computational Complexity*, 12:162–175, 2003. https://link.springer.com/article/10.1007/s00037-003-0181-z.

[27] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010. https://www.cambridge.org/highereducation/books/quantum-computation-and-quantum-information/01E10196D0A682A6AEFFEA52D53BE9AE#overview.

[28] Elijah Pelofske, Andreas Bärtschi, and Stephan Eidenbenz. Quantum Volume in Practice: What Users Can Expect From NISQ Devices. *IEEE Transactions on Quantum Engineering*, 3:1–19, 2022. https://ieeexplore.ieee.org/document/9805433.

[29] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, 2018. https://quantum-journal.org/papers/q-2018-08-06-79/.

[30] William R. Scott. *Group theory*. Dover books on advanced mathematics. Dover Publ., New York, NY, reprint. edition, 1987. https://permalink.catalogplus.tuwien.at/AC01292875.

[31] Peter W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. ArXiv version: https://arxiv.org/pdf/quant-ph/9508027v2.pdf.

[32] Daniel R. Simon. On the Power of Quantum Computation. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, SFCS '94, page 116–123, USA, 1994. IEEE Computer Society. Final version published 1997 in SIAM journal on Computing. https://doi.org/10.1109/SFCS.1994.365701.