

ESTUDIO DE PROPIEDADES METAMÓRFICAS PARA TESTING DE PROGRAMAS CUÁNTICOS

SINHUÉ GARCÍA GIL

GRADO EN MATEMÁTICAS, FACULTAD DE CIENCIAS MATEMÁTICAS
UNIVERSIDAD COMPLUTENSE DE MADRID



TRABAJO FIN DE GRADO

Itinerario de Ciencias de la Computación

Curso 2022-2023

Director:

Luis Fernando Llana Díaz

Madrid, 28 de Junio de 2023

Resumen

Con el gran desarrollo que está aconteciendo en la computación cuántica, tanto en términos de la cantidad de qubits en los sistemas cuánticos como en la fiabilidad de estos, nos acercamos cada vez más a poder hacer un uso real de sus algoritmos. Sin embargo, esto nos genera un nuevo problema o, más bien, unas nuevas inquietudes sobre cómo vamos a programarlos, dado que se vuelven cada vez son más complejos, y también cómo probar su corrección.

Aquí es donde presentamos el *testing* metamórfico como una de las posibilidades que tenemos para alcanzar nuestro objetivo. A lo largo de este documento, introduciremos el estudio de propiedades metamórficas para estos algoritmos, así como las implementaciones y el uso del *testing* metamórfico para la detección de fallos.

Palabras clave: Computación cuántica, qiskit, propiedades metamórficas

Abstract

The quick development that quantum computing has been experiencing in recent years, both in terms of the number of qubits in quantum systems and their reliability, it is opening up the usage of its algorithms. However, this raises new questions, such as how we are going to program or prove the correctness of these algorithms, which are becoming increasingly complex over time.

This is where we introduce metamorphic testing as one of the testing methods to support us in achieving this goal. Throughout this document, we will explore the study of metamorphic rules for these algorithms, as well as the implementation and the use of metamorphic testing for fault detection.

Keywords: Quantum computing, qiskit, metamorphic rules.

Índice general

Índice	1
1. Introducción	2
1.1. Metodología	3
1.2. Objetivos	4
2. Antecedentes	5
2.1. Introducción matemática	5
2.2. Introducción cuántica	8
2.3. Programación cuántica, Qiskit	9
2.3.1. Puertas y circuitos cuánticos	12
2.3.2. Simulaciones y ruido	18
2.4. Propiedades Metamórficas / <i>Testing</i> metamórfico	21
3. Algoritmos cuánticos	23
3.1. Suma	23
3.2. Deutsch	25
3.3. Deutsch-Jozsa	30
3.4. Bernstein-Vazirani	34
3.5. Simon	37
4. Propiedades metamórficas	40
4.1. Deutsch-Jozsa	40
4.2. Bernstein-Vazirani	42
4.3. Simon	46
5. Conclusiones y trabajo futuro	50
5.1. Conclusiones	50
5.2. Trabajo futuro	51
Bibliografía	53

Capítulo 1

Introducción

La memoria y el trabajo presentado a continuación nos mostrarán el recorrido desde los principio básicos de la mecánica cuántica y cómo definen directamente la computación cuántica, hasta una de las posibilidades que tenemos para hacer *testing* sobre estos programas. Antes de continuar con la metodología, veamos donde comenzó todo.

La mecánica cuántica comenzó a desarrollarse en los años 20, pero no sería hasta los años 80 cuando se empezó a plantear la posibilidad de aplicar esta teoría a la computación [1]¹. Paul Benioff presentó en 1980 la máquina de Turing cuántica, que utilizaba la teoría cuántica para describir un ordenador simplificado. En 1984, dicha teoría se utilizó en protocolos criptográficos de la mano de Charles Bennett y Gilles Brassard.

A partir de entonces, empezaron a surgir nuevos algoritmos, principalmente para resolver el problema de oráculo. Entre ellos, se encuentran los algoritmos de Deutsch, Deutsch-Jozsa, Bernstein-Vazirani y Simon. Aquí ya se puede observar la mejora en eficiencia en comparación al ordenador clásico, como Richard Feynman conjeturó en 1982 [2]. Se podría decir que lo que despertó el interés del resto de la comunidad en la importancia que podía tener la computación cuántica ocurrió gracias a Peter Shor en 1994, con sus algoritmos que podrían permitir romper las claves de encriptación RSA y Diffie-Hellman, que aún se utilizan en la actualidad. Desde entonces, la inversión y el interés en el estudio de este campo han ido creciendo, y en 1998 se alcanzó el hito de construir el primer ordenador de dos qubits, lo que hizo factible esta posibilidad.

En la actualidad, IBM tiene el ordenador con mayor número de qubits, con 433 qubits, el `ibm_seattle` presentado a finales de 2022, y están trabajando para presentar el siguiente sistema cuántico con 1121 qubits a finales de este mismo año². Es importante destacar que el sistema anterior solo tenía 127 qubits, ¡pero este se presentó hace solo 2 años, en 2021!

¹https://en.wikipedia.org/wiki/Quantum_mechanics#History

²<https://www.ibm.com/quantum/roadmap>

Con toda esta evolución que está ocurriendo respecto al número de qubits y las mejoras que están realizando para alcanzar mayor fiabilidad, se empieza a abrir la puerta al uso real de los algoritmos y programas cuánticos. Sin embargo, no se alcanzará hasta conseguir ordenadores con mejores características tanto en número de qubits como en precisión. Pero, ¿cómo vamos a comprobar la corrección de estos?

Una de las posibilidades que plantearemos en este trabajo es cómo la unión entre la computación cuántica y el *testing* metamórfico puede ayudarnos a contestar esta pregunta y buscar esa corrección o falta de errores. El *testing* metamórfico es uno de los métodos utilizados para la detección de errores desde su presentación en 1998 [3], y se basa en el estudio de las propiedades necesarias que pueden obtenerse de los algoritmos.

1.1. Metodología

La metodología seguida en este trabajo se puede distinguir en tres fases, las cuales se reflejarán en los siguientes capítulos de esta memoria. El material principal utilizado como base de estudio han sido los libros *Quantum Computation and Quantum Information*, de Michael A. Nielsen y Isaac L. Chuang [4], y *Quantum Computing for Computer Scientists*, de Noson S. Yanofsky y Mirco A. Mannucci [1]. A continuación, vamos a introducir brevemente cada una de estas fases:

- **Antecedentes:** Esta primera fase de estudio, nos enfocamos en avanzar desde los conocimientos básicos adquiridos en el grado, hasta adquirir una base sólida para poder entender los algoritmos cuánticos, la obtención de las propiedades metamórficas y su aplicación en el *testing*. Además de los textos mencionados anteriormente, que han proporcionado resultados más teóricos, en este capítulo se han establecido los conceptos relacionados con el *testing* y propiedades metamórficas, tomando como referencia el artículo *Metamorphic Testing: A Review of Challenges and Opportunities* [5].
- **Programación cuántica y algoritmos:** Una vez adquirida la base necesaria para comprender los algoritmos y la programación cuántica, se llevan a cabo los primeros pasos de programación, empezando por un algoritmo tan sencillo como la suma y la implementación de la misma utilizando Qiskit. A partir de ahí, se fue avanzando algoritmo por algoritmo, desde los más simples hasta llegar a la transformada cuántica de Fourier y aplicaciones. El estudio de estos algoritmos se presentará en el capítulo 3, y todas las implementaciones y pruebas realizadas sobre estos algoritmos, para

no sobrecargar este documento, se encuentran en el repositorio personal de GitHub, <https://github.com/sinugarc/TFG.git> . Además, para el dibujo de circuitos en la memoria se ha utilizado Qcircuit [6].

- **Propiedades y *testing* metamórfico:** Ahora que ya hemos logrado entender e incluso programar nuestros algoritmos cuánticos, es hora de abordar el último eslabón de la cadena: el estudio de las propiedades metamórficas en los algoritmos de Deutsch-Jozsa, Bernstein-Vazirani y Simon. Como apoyo para la comprensión y estudio de esta fase, hemos utilizado como referencia en el artículo *Metamorphic Testing of Oracle Quantum Programs* [7]. Además, esta parte del trabajo se ha realizado en colaboración con Rodrigo de la Nuez Moraleda, y todas las implementaciones realizadas para la *testing* de estos algoritmos se pueden encontrar en el repositorio de GitHub que tenemos en común, <https://github.com/rodelanu/TFG> .

1.2. Objetivos

Una vez analizada la metodología de trabajo y los materiales utilizados, vamos a seguir el mismo esquema para analizar los objetivos. Es importante destacar que el objetivo principal de este trabajo es el estudio de las propiedades metamórficas y su aplicación en forma de *testing* a algoritmos cuánticos. Sin embargo, para llegar a este punto, vamos a ir alcanzando una serie de objetivos y adquiriendo competencias secundarias en cada fase del trabajo.

- **Antecedentes:** Queremos ser capaces de entender las definiciones más básicas, partiendo de los conocimientos matemáticos, especialmente del álgebra lineal. Algunos ya conocidos como que representa una matriz unitaria o hermitiana, así como la relación que existe entre ellas y sus operadores. También se introducirán nuevos conceptos, como el espacio de Hilbert y el producto tensorial, este último nos permitirá generar sistemas más complejos. Además, estudiaremos la parte física de computación cuántica, con sus postulados y su relación con dicha programación. También vamos a introducir los elementos básicos de dicha programación y comprender el sentido general de las simulaciones y ejecuciones de los programas cuánticos.
- **Programación cuántica y algoritmos:** El objetivo principal de este capítulo es claro: queremos ser capaces de crear y analizar programas. Para este fin, nos enfocaremos en entender cómo se han construido los diversos algoritmos y la utilidad que tienen.
- **Propiedades y *testing* metamórfico:** Aquí se desarrollará el objetivo principal del trabajo.

Capítulo 2

Antecedentes

El principal objetivo de este apartado es exponer brevemente al lector las bases, tanto matemáticas como físicas, para comprender y trabajar con propiedades metamórficas, especialmente su aplicación a la computación cuántica. Para lograrlo, haremos un breve repaso a conceptos básicos de álgebra lineal en matemáticas, los postulados de la mecánica cuántica y una introducción a la computación cuántica y el *testing* metamórfico.

Esta sección, que podría ser una simple continuación de la introducción, será más extensa de lo que se podría esperar. Esto se debe a que, para trabajar de forma efectiva el tema a tratar, hemos necesitado adquirir un conjunto de conocimientos considerable.

2.1. Introducción matemática

Para poder desarrollar y comprender la mecánica cuántica, así como la programación cuántica y sus algoritmos, es necesario contar con una base matemática y comprender la notación utilizada en este campo. Aunque las definiciones que se presentarán a continuación pueden parecer aleatorias en este momento, cobrarán sentido a medida que profundicemos en el estudio de la mecánica y la programación cuántica.

Definimos un **espacio de Hilbert**, \mathcal{H} , como un \mathbb{C} -espacio vectorial completo dotado de un producto interno. En particular, vamos a tratar con espacios de Hilbert de dimensión finita, lo cual implica que será completo.

Por el **Teorema de representación de Riesz**, tenemos que \mathcal{H} es anti-isomorfo a \mathcal{H}^* , por ser \mathbb{C} nuestro cuerpo base.

Denotaremos como ket, $|v\rangle$, a un vector v de \mathcal{H} . Análogamente, a toda transformación w de \mathcal{H}^* , la denotaremos como bra, $\langle w|$. Esta notación es conocida como **notación de Dirac** y será utilizada en mecánica cuántica.

Veamos ahora distintas definiciones para operadores:

- Sea $\mathcal{A} : \mathcal{H} \rightarrow \mathcal{H}$ continuo, se denomina **adjunto del operador lineal \mathcal{A}** al único operador $\mathcal{A}^* : \mathcal{H} \rightarrow \mathcal{H}$ tal que $\langle \mathcal{A}v, w \rangle = \langle v, \mathcal{A}^*w \rangle$.
- Se dice que $\mathcal{A} : \mathcal{H} \rightarrow \mathcal{H}$ es un **operador autoadjunto** si $\mathcal{A} = \mathcal{A}^*$. Por lo cual los autovalores de \mathcal{A} son reales.
- Llamaremos matriz **hermitiana** a la matriz A que determina el operador autoadjunto \mathcal{A} . De aquí obtenemos que A^* es la traspuesta conjugada de A .
- Se dice que $\mathcal{U} : \mathcal{H} \rightarrow \mathcal{H}$ continuo, es **unitario** si $\langle v, w \rangle = \langle \mathcal{U}v, \mathcal{U}w \rangle$, que en particular es invertible.

Para finalizar esta sección, veremos una operación que nos permitirá combinar dos espacios vectoriales, en particular, dos espacios de Hilbert. Esta operación se conoce como el producto tensorial y nos permite unir dos sistemas cuánticos [4].¹

Sean \mathcal{H} y \mathcal{H}' dos espacios de Hilbert y \mathcal{B} base de $\mathcal{H} \times \mathcal{H}'$,

$$F(\mathcal{H} \times \mathcal{H}') = \left\{ \sum_{i=1}^n z_i(h_i, h'_i) \mid n \in \mathbb{N}, z_i \in \mathbb{C}, (h_i, h'_i) \in \mathcal{B}(\mathcal{H} \times \mathcal{H}') \right\} \quad (2.1)$$

$$\begin{aligned} \mathcal{R} &= (h_1 + h_2, h') \sim (h_1, h') + (h_2, h') \\ &\quad (h, h'_1 + h'_2) \sim (h, h'_1) + (h, h'_2) \\ &\quad z(h, h') \sim (zh, h') \sim (h, zh') \end{aligned} \quad (2.2)$$

Definimos el **producto tensorial** de \mathcal{H} y \mathcal{H}' , $\mathcal{H} \otimes \mathcal{H}'$, como el espacio cociente entre $F(\mathcal{H} \times \mathcal{H}')$ y \mathcal{R} , es decir, $\mathcal{H} \otimes \mathcal{H}' = F(\mathcal{H} \times \mathcal{H}')/\mathcal{R}$. Partiendo directamente desde la definición de la relación obtenemos:

- **Linealidad:** Sea $z \in \mathbb{C}$, $|h\rangle \in \mathcal{H}$ y $|h'\rangle \in \mathcal{H}'$. Entonces:

$$z(|h\rangle \otimes |h'\rangle) = (z|h\rangle) \otimes |h'\rangle = |h\rangle \otimes (z|h'\rangle) \quad (2.3)$$

¹https://es.wikipedia.org/wiki/Producto_tensorial

■ **Asociatividad:**

- Sea $|h_1\rangle \in \mathcal{H}$, $|h_2\rangle \in \mathcal{H}$ y $|h'\rangle \in \mathcal{H}'$. Entonces:

$$(|h_1\rangle + |h_2\rangle) \otimes |h'\rangle = |h_1\rangle \otimes |h'\rangle + |h_2\rangle \otimes |h'\rangle \quad (2.4)$$

- Sea $|h\rangle \in \mathcal{H}$, $|h'_1\rangle \in \mathcal{H}'$ y $|h'_2\rangle \in \mathcal{H}'$. Entonces:

$$|h\rangle \otimes (|h'_1\rangle + |h'_2\rangle) = |h\rangle \otimes |h'_1\rangle + |h\rangle \otimes |h'_2\rangle \quad (2.5)$$

Además, alcanzamos los siguiente resultados sobre el producto tensorial:

- **Base:** $|i\rangle \otimes |j\rangle$ es una base de $\mathcal{H} \otimes \mathcal{H}'$ donde $|i\rangle$ y $|j\rangle$ pertenecen a una base ortonormal de \mathcal{H} y \mathcal{H}' respectivamente.
- **Producto interno:** los productos internos de \mathcal{H} y \mathcal{H}' inducen naturalmente un producto interno en $\mathcal{H} \otimes \mathcal{H}'$, por lo que hereda la estructura y, con ella, las nociones de adjunta, unitaria, normalidad y hermiticidad.
- **Producto de Kronecker,** se corresponde con el producto tensorial de aplicaciones lineales. Nos será de gran utilidad a lo largo de este trabajo y nos permite visualizar el producto tensorial. Veamos un ejemplo, si A y B son 2 matrices, entonces:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ a_{21}B & a_{22}B & \dots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \dots & a_{mn}B \end{bmatrix} \quad (2.6)$$

Además, si A y B unitarias, entonces $A \otimes B$ es unitaria.

- **Dimensión:** $\dim(\mathcal{H} \otimes \mathcal{H}') = \dim(\mathcal{H}) \cdot \dim(\mathcal{H}')$

Estos dos últimos apartados nos muestran la complejidad que tendremos a la hora de realizar cálculos, ya que la dimensión de las matrices va a crecer de manera exponencial.

2.2. Introducción cuántica

La base principal para el inicio de la computación cuántica fue el desarrollo de la física cuántica. Para comprender mejor las variaciones e implicaciones que tiene, vamos estudiar los postulados de la mecánica cuántica y comprarlos con la mecánica Newtoniana. En este punto, la base matemática presentada en la sección 2.1 empezará a cobrar sentido.

Para empezar, en mecánica clásica, un sistema de N partículas se define mediante un vector en un espacio $\mathbb{R}^{3N} \times \mathbb{R}^{3N}$ donde las primeras coordenadas definen la posición y las últimas la velocidad. La evolución de este sistema se rige por la segunda Ley de Newton, que relaciona la fuerza con la aceleración y la masa.

Por otra parte, en física cuántica, el estado y la evolución de un sistema viene determinado por sus postulados [4][8] que veremos a continuación ², junto con una posible interpretación de cada uno. Estos postulados nos ayudaran posteriormente a establecer la base de nuestros programas cuánticos.

Postulados cinemáticos o de representación:

- **Primer postulado:** El estado en un sistema aislado, en un instante t , se corresponde con $|\varphi(t)\rangle$, en un espacio de Hilbert, \mathcal{H} .
- **Segundo Postulado:** El espacio que representa un sistema compuesto es el producto tensorial de los espacios de cada componente del sistema. Es decir, si tuviéramos n componentes, el espacio factoriza $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2 \otimes \dots \otimes \mathcal{H}_n$.

Postulados dinámicos:

- **Tercer postulado:** Evolución determinista.
 - **Primer apartado:** La evolución de un vector $|\varphi(t)\rangle$ está determinada por la ecuación de Schrödinger, $i\hbar \frac{d|\varphi\rangle}{dt} = H|\varphi\rangle$. Donde H es el Hamiltoniano del sistema, que es un operador hermitiano. Además, se entiende $H(t)$ como un observable asociado a la energía total del sistema.
 - **Segundo apartado:** La evolución de un sistema aislado se describe por una transformación unitaria del estado inicial. $|\varphi(t)\rangle = U(t; t_0)|\varphi(t_0)\rangle$.

²https://en.wikipedia.org/wiki/Mathematical_formulation_of_quantum_mechanics#Postulates_of_quantum_mechanics

- **Cuarto postulado:** Evolución probabilística, tenemos observador.
- **Primer apartado:** Cada medida \mathcal{A} está descrita por un operador hermitiano A que actúa sobre \mathcal{H} , decimos que este operador es un observable, debido a que sus autovectores forman una base de \mathcal{H} . El resultado de medir una cantidad \mathcal{A} debe ser uno de los autovalores correspondientes al observable A .
- **Segundo apartado:** $Prob(\lambda_i) = \frac{\|P_{|v_i\rangle}|\varphi(t)\rangle\|^2}{\|\varphi(t)\rangle\|^2} = \frac{|\langle v_i|\varphi(t)\rangle|^2}{\|\varphi(t)\rangle\|^2}$
- **Tercer apartado:** Si tras realizar una medición \mathcal{A} del estado $|\varphi(t)\rangle$ da como resultado λ_i , entonces el estado del sistema colapsa a la proyección normal de $|\varphi(t)\rangle$ en el subespacio de autovectores asociado a λ_i , $P_{|\lambda_i\rangle}|\varphi(t)\rangle$.

Todos estos postulados van a ser clave en los distintos aspectos de la computación cuántica, como la definición del sistema más simple, el qubit.

2.3. Programación cuántica, Qiskit

La programación cuántica se basa en la creación de un circuito o algoritmo cuántico, generalmente representado de forma geométrica, en el cual se realizan operaciones con operadores unitarios en los diferentes qubits, así como mediciones [1].

Para llevar a cabo nuestros programas cuánticos y simulaciones, nos apoyaremos en Qiskit, que es un paquete de desarrollo libre creado por IBM para la creación y manipulación de programas cuánticos, así como realizar simulaciones³. Ya sea de forma teórica o conectando nuestros programas con los ordenadores cuánticos de IBM. Esto nos proporcionará unos resultados más realistas, donde podremos apreciar el ruido existente en dichos ordenadores en la actualidad.

Se programará Python, debido a que Qiskit es una librería de este lenguaje de programación. Además, para una mejor visualización del código, se utilizará jupyter notebook. Otra opción sería generar los circuitos de forma geométrica en IBM.

Además, existirían otras opciones como *Azure Quantum*⁴ en *C#* de Microsoft, *Cirq*⁵ en *Python* de Google o *Pyket*⁶ también en *Python* entre otros.

³<https://qiskit.org/>

⁴<https://azure.microsoft.com/en-us/services/quantum/>

⁵<https://quantumai.google/cirq>

⁶<https://cqcl.github.io/tket/pytket/api/index.html>

Como mencionamos anteriormente, los postulados cuánticos nos permitirán establecer las bases de la computación cuántica, el primer ejemplo es el qubit.

Si recordamos el postulado 1 de la mecánica cuántica, vamos a definir **qubit** como el sistema cuántico más simple, que va a ser nuestra base en la programación cuántica. Un **qubit** es un espacio bidimensional, donde vamos a suponer que tomamos la base ortonormal $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ y $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. De aquí podemos obtener la combinación lineal de cualquier vector de estado del qubit, aunque los vectores de estado deben de cumplir la condición de normalización, es decir, si $|\varphi\rangle = a|0\rangle + b|1\rangle = \begin{bmatrix} a \\ b \end{bmatrix}$ con $a, b \in \mathbb{C}$, entonces $|a|^2 + |b|^2 = 1$. Esta condición se impone debido a que el módulo al cuadrado de cada coeficiente determina la probabilidad de aparición de ese elemento de la base y por lo tanto, la suma de todas las probabilidades debe ser 1. Además, llamaremos estado de **superposición** a los estados del qubit que se encuentran en el continuo entre $|0\rangle$ y $|1\rangle$.

Estos estados de un qubit se entienden muy bien utilizando la **esfera de Bloch**, que observaremos en la Figura 2.1 ⁷. Debido a que si $|\Psi\rangle = a|0\rangle + b|1\rangle$, $|a|^2 + |b|^2 = 1$. Podemos representar $|\Psi\rangle = e^{i\gamma} \left(\cos \frac{\theta}{2} |0\rangle + e^{i\varphi} \sin \frac{\theta}{2} |1\rangle \right)$, donde $\theta, \varphi, \gamma \in \mathbb{R}$. Si bien es cierto, podemos ignorar el factor $e^{i\gamma}$, ya que no tiene efectos observables. Hay que tener en cuenta que esta representación está limitada a un qubit, porque no hay manera simple de generalizarla.

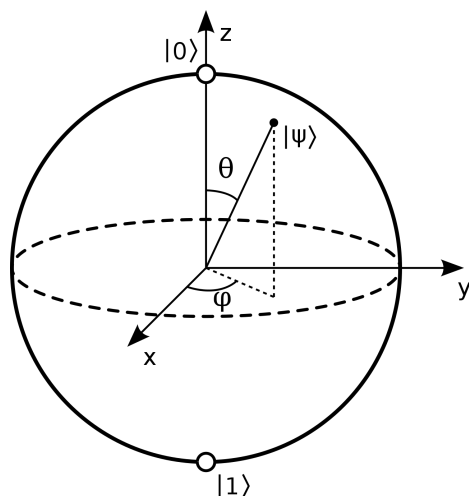


Figura 2.1: Esfera de Bloch

⁷https://en.wikipedia.org/wiki/Bloch_sphere#/media/File:Bloch_sphere.svg

Veamos ahora qué sucede cuando no tenemos un único qubit, sino varios, y la relación que existe entre ellos. Para simplificar, consideremos el caso más simple, un sistema con 2 qubits. Si recordamos el postulado 2, establecía cómo interactúan estos dos qubits, describiendo el sistema compuesto por ambos como el producto tensorial de los dos qubits individuales. Por ello, podemos definir el estado de un sistema de 2 qubits como:

$$|\psi\rangle = a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle \text{ donde } a_{00}, a_{01}, a_{10}, a_{11} \in \mathbb{C}$$

A estos coeficiente complejos se les denomina **amplitud** de cada estado de la base. Al igual que ocurre en el qubit, $|a_{00}|^2 + |a_{01}|^2 + |a_{10}|^2 + |a_{11}|^2 = 1$. Este resultado se puede obtener del producto de Kronecker, en caso de ser un estado producto, y además tendría sentido con la medición de las probabilidades. En este ejemplo se puede ver claramente que la dimensión de la base va a ser exponencial respecto al número de qubits.

Por último y antes de pasar a las puertas cuánticas y cómo operamos en el sistema, vamos a introducir un último concepto, el entrelazamiento cuántico o *entanglement*. Esta es una propiedad de la mecánica cuántica que aparece en un sistema. Por ejemplo, un sistema de 2 qubits están en un estado de *entanglement* cuando no es posible separar el estado como producto tensorial de los dos qubits [1], diremos además que existe una correlación entre ellos. Normalmente se observa a la hora de hacer mediciones en el sistema, también conocido como *entangled measurements*. Si bien es cierto, aún se sigue investigando esta propiedad, así como completar su teoría [4]. Veamos un ejemplo simple que nos permitirá obtener una idea.

Definimos *Bell state* como $|\varphi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$, veamos primero que no se puede expresar como producto tensorial.

$$\sqrt{2}|\varphi\rangle = 1|00\rangle + 0|01\rangle + 0|10\rangle + 1|11\rangle \quad (2.7)$$

Suponemos que $\sqrt{2}|\varphi\rangle = (a_0|0\rangle + a_1|1\rangle) \otimes (a'_0|0\rangle + a'_1|1\rangle)$, entonces:

$$a_0|0\rangle + a_1|1\rangle \otimes (a'_0|0\rangle + a'_1|1\rangle) = a_0a'_0|00\rangle + a_0a'_1|01\rangle + a_1a'_0|10\rangle + a_1a'_1|11\rangle \quad (2.8)$$

Por lo que, $a_0a'_0 = a_1a'_1 = 1$ y $a_0a'_1 = a_1a'_0 = 0$. Pero este sistema de ecuaciones no tiene solución, es decir, $\sqrt{2}|\varphi\rangle$ no se puede expresar como producto tensorial y en particular, el *Bell state*, $|\varphi\rangle$, tampoco.

Ahora queremos realizar una medición con base $\{|0\rangle, |1\rangle\}$, pero sabemos que sólo podemos obtener los resultados $|00\rangle$ o $|11\rangle$. Es decir, cuando realice la medición sobre el primer qubit ya sabremos el valor del segundo sin haberlo medido ⁸. Podemos encontrar otros ejemplos y preguntas muy interesantes sobre *entanglement* en la página de IBM ⁹, donde estudia distintas posibilidades con la puerta CNOT, que presentaremos a continuación, y estados interesantes como *Bell state*, *GHZ states* and *W states*.

Otro resultado interesante es el Teorema de no Clonación, el cual establece que no es posible copiar el estado desconocido de un qubit a otro qubit. Esto se puede entender, debido a que si observáramos el qubit, estaríamos realizando una medición y el estado colapsaría.

Teorema 2.1 (No clonación) [4]: Sean \mathcal{H} un espacio de Hilbert en \mathbb{C} y $|\psi\rangle \in \mathcal{H}$. Entonces, $\nexists U$ unitario sobre $\mathcal{H} \otimes \mathcal{H}$ tal que $U|\varphi\rangle|\psi\rangle = |\varphi\rangle|\varphi\rangle \quad \forall |\varphi\rangle \in \mathcal{H}$ ¹⁰.

2.3.1. Puertas y circuitos cuánticos

Partiendo del tercer postulado, en particular el apartado 2, podemos observar que existe una correspondencia entre el Hamiltoniano H , por ser hermitiano, y un operador unitario U . Estos operadores unitarios serán nuestras **puertas cuánticas** que utilizaremos junto a los qubits. Este postulado viene de la evolución **determinista** desde un puesto de vista dinámico, donde no se realiza ninguna observación sobre el sistema. Uno de los puntos importantes de que estos operadores sean unitarios es que conservan la norma, lo cual garantiza que no se rompa la condición de normalización en la definición de qubit.

En programación cuántica, es posible crear operadores unitarios directamente utilizando matrices que cumplan las condiciones necesarias. Aunque, se suelen utilizar puertas cuánticas específicas que están diseñadas para utilizarse en ordenadores cuánticos reales [8]. A continuación, veremos algunas de las puertas cuánticas más útiles para un qubit [4] ¹¹:

- **Puertas de Pauli:** Estas puertas son la más básicas y nos van a permitir, a excepción de la identidad, realizar rotaciones de π radianes dentro de la esfera de Bloch, cada una sobre el eje que indica su propio nombre. Las matrices que determinan estas puertas generan una base del espacio de matrices hermitianas 2×2 .

⁸<https://jcc.dcc.fceia.unr.edu.ar/2006/slides/2006-diazcaro-samborskiforlese.pdf>

⁹<https://quantum-computing.ibm.com/composer/docs/ibmqx/guide/entanglement>

¹⁰https://es.wikipedia.org/wiki/Teorema_de_no_clonaci%C3%B3n

¹¹https://en.wikipedia.org/wiki/Quantum_logic_gate

- **Puerta identidad:** $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, con puerta geométrica $\text{---}\boxed{I}\text{---}$
- **Y:** La puerta Y , $\text{---}\boxed{Y}\text{---}$, viene determinada por la matriz $Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
- **Z:** La puerta Z , $\text{---}\boxed{Z}\text{---}$, viene determinada por la matriz $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
- **X:** La puerta X es un operador que viene determinado por la matriz $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

Esta puerta sería la análoga cuántica a la puerta NOT clásica y nos permite

$$|0\rangle \rightarrow |1\rangle, |1\rangle \rightarrow |0\rangle, \text{ por lo que dado } |\varphi\rangle = a|0\rangle + b|1\rangle \Rightarrow X|\varphi\rangle = b|0\rangle + a|1\rangle$$

Su puerta geométrica al realizar los circuitos será: $\text{---}\boxed{X}\text{---}$

- **Puerta de Hadamard, H :** Este operador viene determinado por $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

Probablemente la puerta más interesante de todas, que nos permite poner el qubit en un estado especial, es más, dichas transformaciones sobre la base $\{|0\rangle, |1\rangle\}$ tiene su propia notación y forman una base interesante:

$$|+\rangle = H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \quad |-\rangle = H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

A su vez, al igual que las matrices de Pauli, la matriz de Hadamard realiza una rotación de π radianes, pero esta vez sobre el eje $(\hat{x} + \hat{z})/\sqrt{2}$. Puerta: $\text{---}\boxed{H}\text{---}$

- **Puerta de cambio de fase, $P(\theta)$:** Esta puerta nos va a permitir realizar rotaciones de un ángulo θ sobre el eje \hat{z} . Y está determinada por $P(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$. El nombre de esta puerta puede llevar a confusión, ya que mencionamos anteriormente que la fase no era observable, pero en ese caso nos referíamos a la fase global del qubit. En este caso el cambio de fase se realiza sobre los ejes.

Casos particulares de puertas importantes:

- $P(\pi) = Z$.
- $P(\pi/2) = S$, también conocida como puerta de fase, $\text{---}\boxed{S}\text{---}$

- $P(\pi/4) = T, \text{---}\boxed{T}\text{---}$

Análogamente, se puede definir las matrices de rotación para los distintos ejes. Y en general, vamos a definir la puerta de rotación sobre un eje cualquiera \hat{n} , esta puerta queda determinada por $R_{\hat{n}}(\theta) = \cos\left(\frac{\theta}{2}\right) I - i \sin\left(\frac{\theta}{2}\right) (n_x X + n_y Y + n_z Z)$

La idea de introducir al lector con esta puerta de rotación, además de su utilidad, se debe a que nos va a permitir presentar el siguiente teorema. Como se mencionó anteriormente, cualquier matriz unitaria 2 x 2, puede definir un operador sobre un qubit, veamos que relación hay con las rotaciones.

Teorema 2.2: Descomposición Z-Y para un único qubit [4]: Sea U un operador unitario sobre un qubit. Entonces, existen números reales α, β, γ y δ tal que:

$$U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta)$$

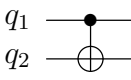
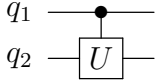
Existe un resultado análogo para X-Y. Este teorema va a permitir descomponer cualquier operador unitario en estas rotaciones y así, los ordenadores cuánticos actuales, pueden procesar cualquier circuito independientemente de las diferencias entre puertas que componen el circuito cuántico y las puertas que dispone el ordenador [8].


Observemos ahora las puertas más importantes para más de un qubit, en particular nos vamos a fijar en 2 y 3 qubits, para 2 utilizaremos la base $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ y la análoga para 3 qubits.

- **CNOT**, también conocida como **puerta de control Pauli-X**. Es un caso particular de las puertas de control sobre cualquier operador. En este caso, sobre la puerta X. Este operador viene determinado por la matriz,

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & X \end{bmatrix}$$

La interpretación de esta puerta es que si el primer qubit es $|1\rangle$, realiza la operación X sobre el segundo qubit. En general, podríamos definir la puerta controlada para un operador U , de forma análoga intercambiando X por U .

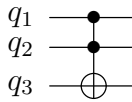
La representación de CNOT en un circuito es:  . En general, .

- **SWAP**, o puerta de intercambio de qubits. Con notación  . La determina,

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Hay que aclarar que esta puerta no copia los estados de los qubits, sino que los intercambia. Por lo que no contradice el teorema de no clonación introducido anteriormente.

- **Toffoli, CCNOT**. Puerta de control sobre 2 qubits q_1, q_2 aplicando la puerta X a q_3 .

Usaremos como notación  y queda determinada por,

$$CCNOT = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & X \end{bmatrix}$$

Esta puerta, al igual que CNOT, podría generalizarse para un operador unitario U.

Existe un concepto que se utiliza a menudo que es el de **conjunto de puertas universales** ¹², que se define como el conjunto de puertas al que toda operación unitaria se puede reducir. Aunque, esto teóricamente es imposible debido a que el número de puertas que se pueden construir es no numerable, y cualquier conjunto finito que definamos sí lo es. Para solucionar este problema, nos apoyamos en el teorema de *Solovay-Kitaev* ¹³, que garantiza que esta reducción se puede hacer de forma eficiente, es decir, aproximándola con la precisión necesaria. Los ejemplos más conocidos son:

- $\{R_{\hat{x}}, R_{\hat{y}}, R_{\hat{z}}, P, CNOT\}$
- $\{CCNOT, H\}$

Esto se debe a que existen diversas relaciones entre las puertas que hemos estudiado, como por ejemplo:


¹²https://en.wikipedia.org/wiki/Quantum_logic_gate#Universal_quantum_gates

¹³https://en.wikipedia.org/wiki/Solovay%E2%80%93Kitaev_theorem

- $ZX = iY = -XZ$
- $CNOT = \exp\left(i\frac{\pi}{4}(I - Z_1)(I - X_3)\right)$
- $HZH = X$
- $SWAP = \frac{I \otimes I + X \otimes X + Y \otimes Y + Z \otimes Z}{2}$
- $CCNOT = \exp\left(i\frac{\pi}{8}(I - Z_1)(I - Z_2)(I - X_3)\right) = \exp\left(-i\frac{\pi}{8}(I - Z_1)(I - Z_2)(I - X_3)\right)$

Los subíndices indican a que qubit se aplica en caso de ser necesario. Alguna de estas relaciones serán observadas al implementar los algoritmos.

Ahora bien, hasta este punto no hemos observado el sistema, simplemente hemos llevado a cabo operaciones unitarias, es decir, aún estamos en un sistema determinista sin tener un conocimiento real de lo que está sucediendo. Si recordamos el cuarto postulado de la física cuántica, se refiere a la evolución dinámica del sistema cuando era observado. Aquí dejamos atrás la evolución determinista y pasamos a la probabilística. Para comprender que ha ocurrido en nuestro sistema y obtener resultados, vamos a necesitar realizar mediciones.

Esta será la última operación que presentar, , la **medición** de un qubit. Si bien es cierto que podemos medir sobre distintas base, vamos a tomar como referencia $\{|0\rangle, |1\rangle\}$.

Pero, ¿qué va a ser realmente la medición de un qubit?

Esta medición viene totalmente determinada por el postulado 4, se obtendrá uno de los elementos donde proyectamos con cierta probabilidad. Interpretando la fórmula vista anteriormente, esta probabilidad va a ser el cuadrado de su amplitud. Es decir, si $|\varphi\rangle = a|0\rangle + b|1\rangle$ es el estado de un qubit antes de la medición, la probabilidad de que el resultado obtenido sea $|0\rangle$ es $|a|^2$ y para $|1\rangle$ es $|b|^2$. Lo que hay que tener en cuenta, es que una vez realizada una medición, apartado 3, hemos modificado el qubit y a partir de entonces $|\varphi\rangle = |0\rangle$ o $|\varphi\rangle = |1\rangle$.

Para guardar esta información utilizaremos bits clásicos, en la Figura 2.2 se observa como desde la puerta de medición hay una caída hacia el bit clásico donde se almacena.

Con todas estas operaciones, nos ayuda a poner los qubits en los distintos estados queridos y permitir realizar nuestros programas obteniendo una medición final. Al fin y al cabo,

un programa cuántico es una sucesión de operaciones aplicadas sobre los qubits del sistema.

Así que permítanme mostrarles un ejemplo que será utilizado y explicado en el capítulo siguiente. Este ejemplo, Figura 2.2, es la implementación del algoritmo de Bernstein-Vazirani, en adelante BV, para una cadena s de longitud 4. Lo que debe hacer el algoritmo es descubrir esa cadena.

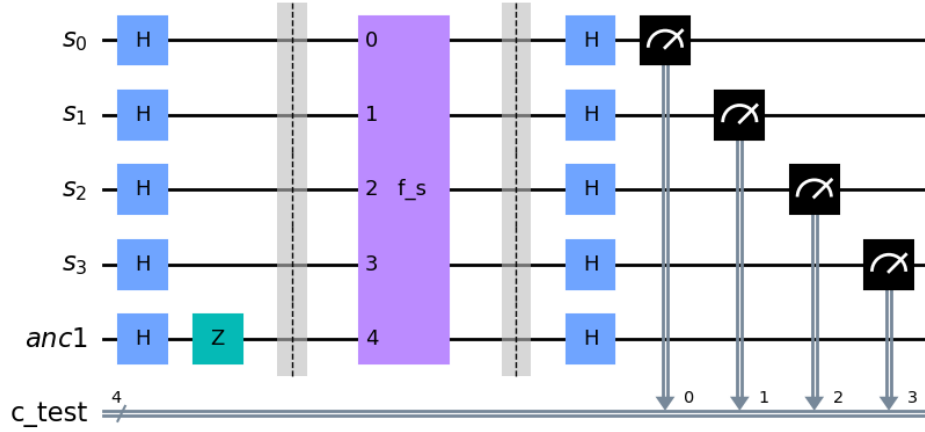


Figura 2.2: Circuito, algoritmo BV para s de longitud 4

Detallemos brevemente los elementos principales de este circuito:

- **Qubits:** Se representan en las primeras líneas, cada uno en una línea distinta. El circuito de la figura 2.2 se compone de 5 qubits. Entre ellos vemos un qubit llamado *anc1*, que es un qubit ancilla con un estado inicial conocido. Algunas veces este qubit auxiliar nos servirá como apoyo para realizar operaciones o almacenará el resultado. Se debería seguir el siguiente esquema para que sea invertible si vemos U como oráculo:

$$\begin{array}{ccc} |x\rangle & \text{---} \boxed{U} \text{---} & |x\rangle \\ |y\rangle & \text{---} \boxed{U} \text{---} & |y \oplus U_x\rangle \end{array}$$

- **Bits clásicos:** Se representan en la última fila, todos juntos. Podemos ver el número, en este caso 4, que nos indica el número de bits clásicos que tenemos.
- **Puertas para un qubit:** Podemos observar puertas ya conocidas como Hadamard, Z o la medición que vemos como cae hacia el cable de bits clásicos indicando en qué bit se registra la medición.

- **Puerta f_s :** Como ya mencionamos anteriormente puede haber puertas que se apliquen a varios qubits, en este caso tenemos la puerta, o bloque, que replica la función del problema de Bernstein-Vazirani. Dentro de este bloque tenemos puertas más pequeñas como las presentadas anteriormente. En particular, f_s tiene puertas CNOT. En diversos algoritmos, se representa esta puerta como un bloque debido a que la podemos entender como un oráculo, del que no sabemos como funciona internamente.
- **Barreras:** Nos sirven como apoyo para visualizar el algoritmo y poder separar en secciones.

2.3.2. Simulaciones y ruido

Una vez que ya hemos visto las puertas que podemos usar en un circuito cuántico, así como un ejemplo del mismo, veamos una introducción a las distintas opciones para la ejecución. *Qiskit* traduce a bajo nivel a *qasm*, que es lo que se va a ejecutar al final. Para ello tenemos las dos opciones siguientes ¹⁴:

- **Simulación**, en nuestro caso con los simuladores de IBM. Esta posibilidad nos va a ofrecer una simulación teórica de lo que ocurre en nuestro circuito. Será ejecutado a través de un simulador de IBM y nos va a permitir obtener resultados con seguridad y sin ningún problema asociado de errores o ruido. Hay que entender que las simulaciones van a tener una limitación debido a la dimensión de las matrices con las que estamos tratando, ya que el coste es exponencial.
- **Ordenador cuántico**, utilizaremos los ordenadores de IBM. La tecnología para la creación de ordenadores cuánticos más fieles sigue avanzando. Esta intenta mitigar los errores que tiene cada operación, así como los errores relacionados con el ruido del entorno que influyen en el estado del sistema modificándolo. Cada ordenador tiene su propia estructura y sus estudios sobre los errores que se cometen en cada qubit, así como calibraciones. Se puede ver en la figura 2.3 la estructura del ordenador y los errores de cada qubit, además de observar que no todos los qubits tienen relación entre sí, por ejemplo, los qubit 2 y 15 no tienen relación. *Qiskit* analizará el circuito que le proponemos y lo adaptará a la arquitectura del sistema elegido para la ejecución, con el objetivo de minimizar los errores cometidos.

¹⁴https://qiskit.org/documentation/qc_intro.html

Ambas opciones utilizan el mismo mecanismo, repiten el proceso tantas veces como les sea requerido y muestran la frecuencia acumulada de los resultados (mediciones) o directamente las probabilidades sobre el número total de ejecuciones.

Veamos un ejemplo para entender la diferencia de resultados de utilizar un opción u otra. Para ello vamos a usar los resultados del circuito presentado en la figura 2.2 del algoritmo de Bernstein-Vazirani que se presentará en el siguiente capítulo. Este algoritmo nos debería dar un resultado único, pero veamos qué ocurre.

Se puede observar en la Figura 2.4 la diferencia entre la simulación teórica (a) y la ejecución con el sistema `ibmq_lima` (b). Nuestra simulación nos ha dado únicamente el resultado querido, pero al ejecutarlo en un sistema cuántico nos ha dado una variedad de resultados, aunque no todos los posibles. Si bien es cierto que el más probable, con suficiente diferencia, es idéntico al obtenido con la simulación. Cabe destacar de la Figura 2.4b que los de mayor probabilidad, una vez eliminada la solución, son aquellos en los cuales solo varía 1 dígito y por el contrario, el complementario binario no ha aparecido como resultado en ninguna de las ejecuciones realizadas.

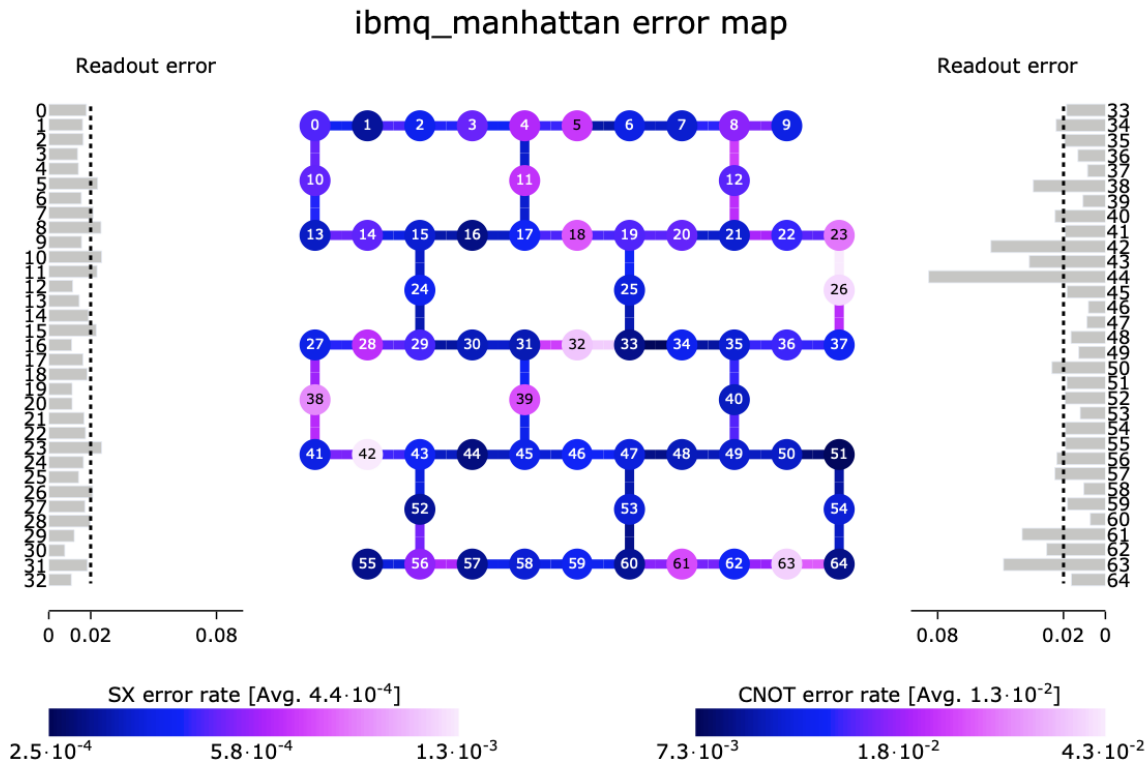


Figura 2.3: Mapa de errores del `ibmq_manhattan`

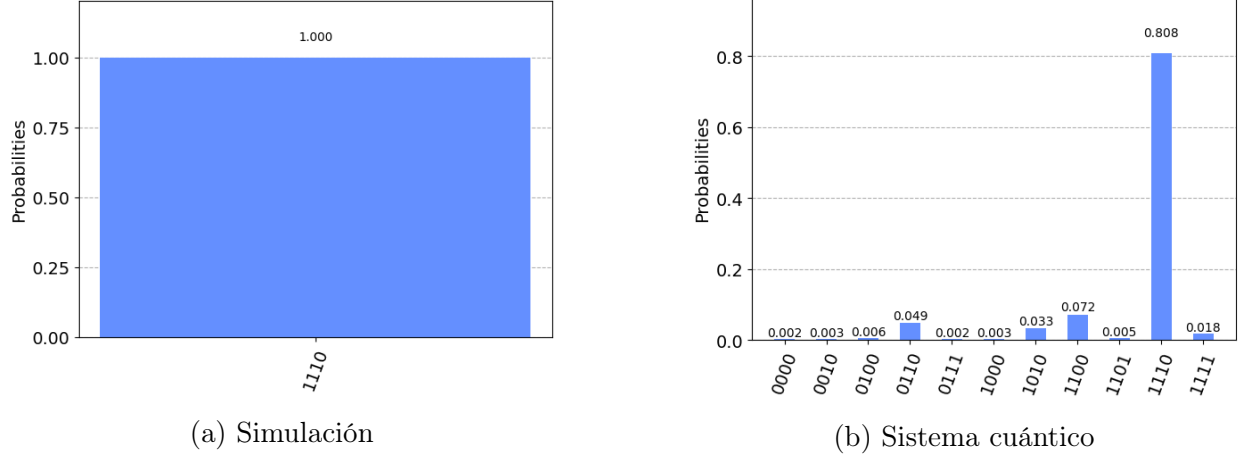


Figura 2.4: Diferencias en resultados de ejecución del algoritmo de Bernstein-Vazirani

Para acabar con esta sección, como curiosidad y por completitud, hemos mencionado antes que *Qiskit* traduce a *qasm* a la hora de ejecutar. La función utilizada en *Qiskit* para ensamblar el programa es *assemble* que devuelve un objeto de clase *QasmQobj*¹⁵. El código que utilizaría *Qiskit* para la ejecución del circuito presentado anteriormente lo podemos encontrar en el trabajo generado en el sistema cuántico de IBM¹⁶. Además, vamos a poder observar en la Figura 2.5 el circuito transformado acorde a las puertas de las que dispone el sistema cuántico elegido. Este circuito es el análogo a la Figura 2.2 con resultados en la Figura 2.4b.

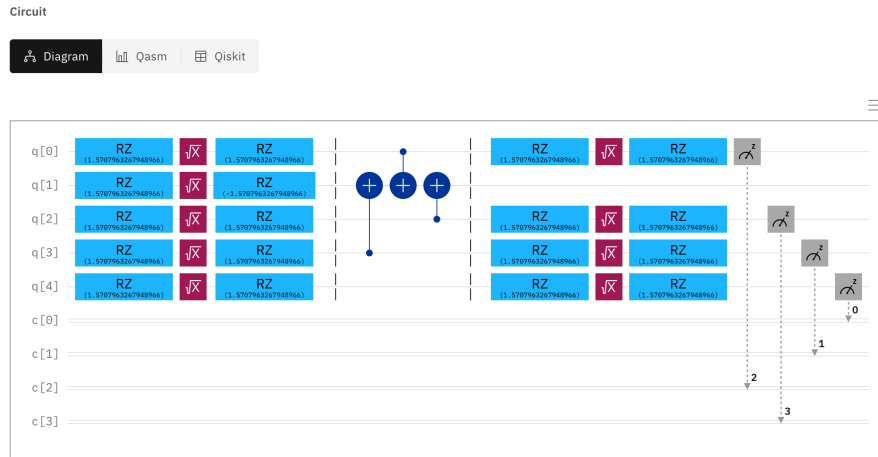


Figura 2.5: Circuito del sistema ibmq_lima para BV, Figura 2.2

¹⁵<https://qiskit.org/documentation/stubs/qiskit.compiler.assemble.html>

¹⁶<https://quantum-computing.ibm.com/>

2.4. Propiedades Metamórficas / *Testing* metamórfico

Informalmente entendemos como propiedad metamórfica, en adelante MR del inglés *metamorphic rule*, aquella que podemos derivar de forma lógica de una definición o especificación. Empecemos con un ejemplo para ponernos en situación, nos vamos a fijar en la función seno, $f(x) = \sin(x)$.

La definición que aprendemos cuando se empieza a ver trigonometría, es que el seno es la proporción entre el cateto opuesto y la hipotenusa en un triángulo rectángulo. Teniendo esta imagen la cabeza es muy fácil darse cuenta que $\sin(x) = \sin(x + 2\pi) = \sin(\pi - x)$. Estas son dos propiedades metamórficas de la función seno.

Veamos ahora que es lo que consideramos formalmente una MR y como llegamos a los pasos de *testing* metamórfico, en adelante MT del inglés *metamorphic testing* [5]:

- **Relación metamórfica**, MR: Sea $f : X \rightarrow Y$ una función. Se considera que $\mathcal{R} \subseteq X^n \times Y^n$ es una **regla metamórfica** si es una relación entre una secuencia de entrada $\langle x_1, x_2, \dots, x_n \rangle$ con $n > 1$ y sus salidas correspondientes $\langle f(x_1), f(x_2), \dots, f(x_n) \rangle$. Es decir, es una propiedad necesaria de f .
- **Source/follow-up input**: Sea \mathcal{R} una relación metamórfica y sea $\langle x_1, x_2, \dots, x_k \rangle$ la secuencia original con sus respectivos resultados. Denotaremos como **source input** a $\langle x_1, x_2, \dots, x_k \rangle$ los cuales son datos definidos o caracterizados, es decir, ya conocidos. A su vez, podemos generar $\langle x_{k+1}, x_{k+2}, \dots, x_n \rangle$, los cuales están contruidos en base a la entrada original e incluso a la salida de esta. A esta secuencia la llamaremos **follow-up input**.
- **Grupo metamórfico de entrada**: Llamaremos **grupo metamórfico de entrada** a la secuencia definida por *source* y *follow-up input*, es decir, $\langle x_1, x_2, \dots, x_k, x_{k+1}, \dots, x_n \rangle$.
- **Testing metamórfico**, MT: Sea f la función objetivo, P una implementación de f y \mathcal{R} una regla metamórfica de f .

Para realizar **testing metamórfico** sobre P seguiremos los siguiente pasos:

- Definimos \mathcal{R}' reemplazando f por P en \mathcal{R} .
- Dado el *source input*, generamos sus salidas según P , construimos a partir de estos el *follow-up input* $\langle x_{k+1}, \dots, x_n \rangle$, y obtenemos $\langle P(x_{k+1}), \dots, P(x_n) \rangle$.

- Estudiamos los resultados obtenidos respecto a \mathcal{R}' . Si \mathcal{R}' no se satisface entonces podemos afirmar que P no es correcto.

La estrategia presentada anteriormente para MT, será la que se siga en la implementación de las propiedades que se obtendrán a lo largo de este documento.

Para finalizar con esta introducción vamos a revisar brevemente un par de ventajas y retos que presenta el camino que estamos tomando para probar la corrección, o más bien la no corrección, de un algoritmo.

Ventajas:

- **Simplicidad.** El concepto e interpretación de una MR es bastante simple en comparación con otros conceptos que se utilizan dentro del campo del *testing*. Se ha visto en estudios, que incluso gente con poca experiencia, podrían utilizar estas técnicas en relativamente poco tiempo de forma efectiva [5][9][10].
- **Facilidad en la implementación.** Si partimos de la ventaja anterior como es la simplicidad de la definición, podemos continuar que la implementación de este test es prácticamente seguir los pasos explicados en la definición de MT.

Retos:

- **Generación efectiva de los grupos metamórficos de entrada.** Aún se está estudiando qué garantías tenemos en la efectividad que puede tener la elección del grupo metamórfico de entrada en la demostración de la corrección de un algoritmo, y en particular, la forma en la que obtenemos nuestro *source input*, ya que *follow-up input* se genera a partir de esta. Al fin y al cabo, nuestro objetivo es maximizar la identificación de errores o defectos en P .
- **Estructura del MT.** Debido a la gran variedad de MR, aún no hay un acuerdo sobre una estructura definida y formal que nos proporcione seguridad en nuestras pruebas y englobe todas las posibilidades que tenemos dentro de las MR. Sin embargo, es importante destacar que ha sido útil para identificar diversos errores en sistemas muy estudiados con otros métodos de *testing*, como por ejemplo los compiladores GCC y LLVM de C, en los cuales se encontraron más de 100 fallos [5][11][12][13].

Los retos de MT pueden ser una buena base para todo el trabajo futuro que se puede realizar en este campo y las posibilidades que este nos puede ofrecer.

Capítulo 3

Algoritmos cuánticos

El siguiente paso en este camino hacia la unión entre la computación cuántica y el *testing* metamórfico es la creación de algoritmos o programas cuánticos que nos ayuden a resolver problemas propuestos.

Todo el desarrollo de estos algoritmos se pueden encontrar en los libros principales [4][1]. A continuación, presentaremos todos los algoritmos finales que nos permitan alcanzar nuestros objetivos. Sin embargo, solo vamos a mostrar el proceso completo de creación del algoritmo de Deutsch por su simplicidad. Esto se debe a que, para alcanzar nuestro objetivo, debemos realizar modificaciones y cálculos en nuestros algoritmos hasta encontrar la combinación correcta de puertas que nos permita resolverlo.

Esta sección seguirá prácticamente la misma estructura para cada apartado. Comenzará con una introducción, seguida de la exposición del problema a resolver. Luego presentaremos el algoritmo que resuelve el problema y su creación en el caso de Deutsch, además de las pruebas realizadas. Es importante recordar que toda la programación realizada con estos algoritmos se encuentra en el repositorio de GitHub <https://github.com/sinugarc/TFG.git>

3.1. Suma

Este primer algoritmo nos va a servir como primera toma de contacto con la programación cuántica, las puertas que podemos utilizar y el uso de *Qiskit*. Veamos cual es el problema a resolver:

Problema 1: Dadas dos cadenas binarias de igual longitud, queremos obtener su suma bit a bit. Esta operación se utilizará posteriormente y se toma \oplus como notación.

La implementación de este problema es muy simple, bastará con incluir tantos qubit ancilla como longitud tenga la cadena para almacenar el resultado. De esta manera nos aseguramos que no se modifiquen los valores originales de entrada. Bastará con utilizar la puerta $CNOT$ para ir modificando los valores ancilla correspondientes, dando lugar en los valores ancilla al resultado de la suma bit a bit de ambas cadenas.

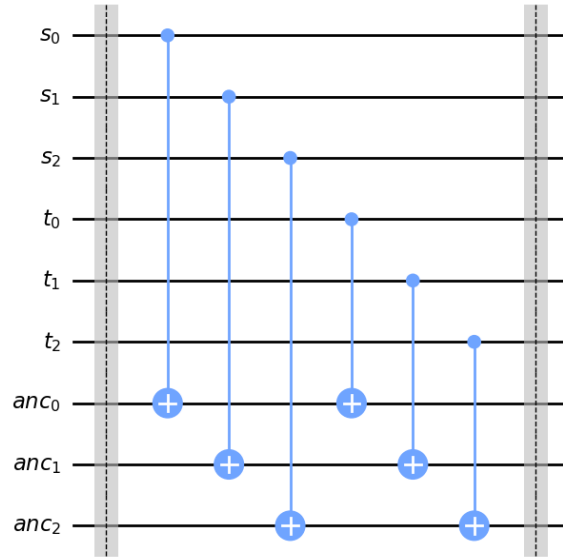


Figura 3.1: Suma binaria bit a bit, para cadenas s y t de longitud 3

Solo se ha representado la parte en la que se realiza la suma sin definir s o t y la medición a realizar.

Problema 2: Dadas dos cadenas binarias, queremos obtener la suma de ambas, interpretando estas cadenas como si fueran la representación de dos números naturales.

El problema ahora se complica un poco más, ya que cuando realizamos la suma de números naturales tenemos que ser capaces de programar si necesitamos llevarnos 1 o no. Para solucionarlo vamos a utilizar la puerta $CCNOT$, que nos va a permitir modificar el siguiente qubit ancilla si fuera necesario. De esta manera nos aseguramos de realizar la suma correctamente junto al uso de la puerta $CNOT$.

Problema: Dada una función $f : \{0, 1\} \rightarrow \{0, 1\}$ balanceada o constante, la cual no podemos observar su definición. Queremos determinar si esta función es constante o balanceada.

Solución clásica: Tenemos que evaluar f en ambos valores y comparar las soluciones.

Veamos ahora que podemos hacer con un ordenador cuántico, ¿seremos capaces de evaluar f una única vez?

Primero observamos un ejemplo particular y cómo vamos a llevarlo a un circuito cuántico. Sea $f(0) = 1$ y $f(1) = 0$, buscamos una matriz unitaria que nos permita representarla en un circuito cuántico, aunque tendremos que hacer alguna modificación más. Por ahora lo que obtenemos es,

$$\begin{array}{c} \mathbf{0} \quad \mathbf{1} \\ \mathbf{0} \quad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ \mathbf{1} \end{array}$$

Donde la columna representa la entrada y la fila la salida. Como ya se mencionó al explicar los qubits del circuito (figura 2.2), es importante conservar la entrada y por lo tanto si U_f es la caja negra que representa a f , nuestro circuito será:

$$\begin{array}{ccc} |x\rangle & \text{---} & |x\rangle \\ |y\rangle & \text{---} & |y \oplus f(x)\rangle \end{array} \quad U_f$$

Es más, si aplicáramos U_f dos veces obtendríamos la entrada:

$$\begin{array}{ccccc} & & |x\rangle & & \\ |x\rangle & \text{---} & U_f & \text{---} & |x\rangle \\ |y\rangle & \text{---} & U_f & \text{---} & |y\rangle \end{array}$$

$|y \oplus f(x)\rangle$

Esto se debe a que $f(x) \oplus f(x) = 0$. Pero claro, ¿qué matriz realmente representa a U_f ? Veamos como quedaría respecto a la base:

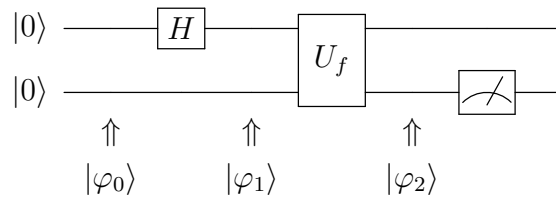
$$\begin{array}{c}
|00\rangle \quad |01\rangle \quad |10\rangle \quad |11\rangle \\
\begin{array}{c} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{array} \left[\begin{array}{cccc} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right]
\end{array}$$

Se podría comprobar que esta matriz es su propia adjunta, por lo que es invertible y unitaria. Es decir, cumple las características necesarias para ser una puerta de un circuito cuántico.

Ahora que ya tenemos la caja negra deseada, determinada por una matriz unitaria, nos queremos poner a resolver el problema. Como comenté en la introducción de este capítulo, este será el único algoritmo al que le seguiremos la pista de razonamiento de principio a final, es decir, partiremos de una idea inicial y acabaremos en el algoritmo de Deutsch que resuelve este problema.

Nuestro objetivo es mejorar la complejidad de la solución del problema respecto a la solución clásica. Recordamos que necesitamos evaluar f dos veces, por lo que nuestro objetivo va a ser obtener el resultado evaluando f una única vez. Para ello nos vamos a sustentar en la superposición, para así analizar que ocurre en el $|0\rangle$ y en el $|1\rangle$ al mismo tiempo, es decir, vamos a tener que utilizar la puerta de Hadamard.

Primer intento: Tomamos nuestro primer *input* para el problema, por ejemplo $|x\rangle = |0\rangle$ e $|y\rangle = |0\rangle$ y creamos el primer circuito:



Donde cada φ_i va a representar el estado del sistema en cada momento, esto nos va a ayudar a analizar de forma determinista lo que ocurre en nuestro circuito.

Podemos resumir el circuito como $U_f(H \otimes I)|0,0\rangle$, veamos ahora que estados tenemos en cada instante i :

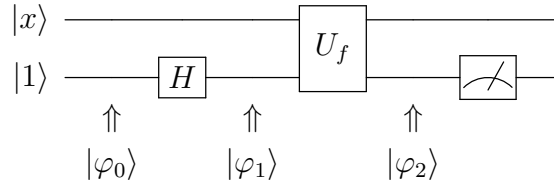
- $|\varphi_0\rangle = |0\rangle \otimes |0\rangle = |0, 0\rangle$
- $|\varphi_1\rangle = (H \otimes I)|0, 0\rangle = \left[\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right] |0\rangle = \frac{|0, 0\rangle + |1, 0\rangle}{\sqrt{2}}$
- $|\varphi_2\rangle = \frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}}$

Si nos fijamos en el ejemplo que expuesto al principio del algoritmo obtendríamos el siguiente estado antes de la medición:

$$|\varphi_2\rangle = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \frac{|0, 1\rangle + |1, 0\rangle}{\sqrt{2}} \quad (3.1)$$

De aquí podemos observar, que sin importar donde hagamos la medición, el resultado no va a ser concluyente, porque vamos a tener una probabilidad de 0.5 de obtener $|0\rangle$ o $|1\rangle$. Es decir, este primer intento no nos sirve para nuestro objetivo.

Segundo intento: Veamos ahora que ocurre si en vez de poner en superposición el primer qubit, ahora ponemos el segundo, tomando $|1\rangle$ como *input*:



Ahora iremos directamente a $|\varphi_2\rangle$:

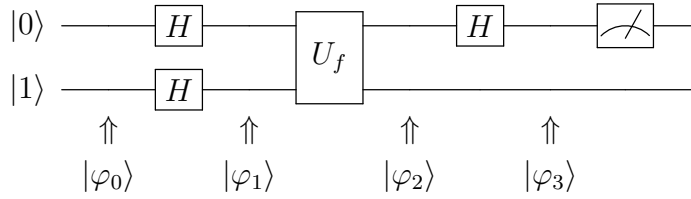
$$|\varphi_2\rangle = |x\rangle \left[\frac{|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} \right] = \begin{cases} |x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] & \text{si } f(x) = 0 \\ -|x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] & \text{si } f(x) = 1 \end{cases} \quad (3.2)$$

Esto lo podemos resumir como:

$$|\varphi_2\rangle = (-1)^{f(x)} |x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \quad (3.3)$$

Pero al igual que en intento anterior, al intentar medir en cualquiera de ambos qubits no obtendríamos ningún resultado concluyente. Esto nos va a llevar al último intento.

Tercer intento, Algoritmo de Deutsch: Al no ser capaces de obtener un resultado poniendo sólo un qubit en superposición, esta vez ponemos ambos en estado de superposición, con el *input* $|0, 1\rangle$. Además, medimos sobre el qubit superior tras aplicar una puerta de Hadamard, que recordamos es su propia inversa. Este es el circuito final que representa al algoritmo de Deutsch:



Esto matricialmente nos queda como $(H \otimes I) U_f (H \otimes H) |0, 1\rangle$.

Analizamos ahora todos los estados $|\varphi_i\rangle$:

- $|\varphi_0\rangle = |0\rangle \otimes |1\rangle = |0, 1\rangle$
- $|\varphi_1\rangle = \left[\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]$
- $|\varphi_2\rangle = \left[\frac{(-1)^{f(0)} |0\rangle + (-1)^{f(1)} |1\rangle}{\sqrt{2}} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]$

Hemos obtenido este resultado substituyendo $|x\rangle$ en la ecuación 3.3. Pero veamos que ocurre separando $(-1)^{f(0)} |0\rangle + (-1)^{f(1)} |1\rangle$ según nuestras posibilidades:

- f es constante: $\frac{(-1)^{f(0)} |0\rangle + (-1)^{f(1)} |1\rangle}{\sqrt{2}} = (\pm 1) \left[\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right]$
- f es balanceada: $\frac{(-1)^{f(0)} |0\rangle + (-1)^{f(1)} |1\rangle}{\sqrt{2}} = (\pm 1) \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]$

- Y por último obtenemos:

$$|\varphi_3\rangle = \begin{cases} (\pm 1) |0\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] & \text{si } f(x) \text{ es constante} \\ (\pm 1) |1\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] & \text{si } f(x) \text{ es balanceada} \end{cases} \quad (3.4)$$

Por lo que hemos conseguido nuestro objetivo. Tras una única evaluación de f según la medición del primer qubit, sabremos si f es constante con medición $|0\rangle$ o balanceada, $|1\rangle$. Consiguiendo así nuestro objetivo.

Oráculo: El oráculo se crea como operador a partir de matriz que representa la función.

Simulaciones: Las simulaciones y código se encuentran en el repositorio ².

3.3. Deutsch-Jozsa

El algoritmo de Deutsch-Jozsa [1] es una generalización del algoritmo de Deutsch del apartado anterior, pero ahora trabajamos con $f : \{0,1\}^n \rightarrow \{0,1\}$. El dominio se puede entender como los números escrito en binario de 0 a $2^n - 1$. En este caso vamos a redefinir los conceptos anteriores:

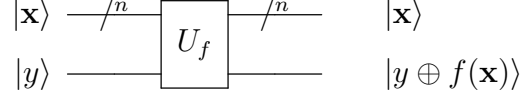
- Diremos que f es **constante** si todo el dominio va a 0 ó a 1.
- Diremos que f es **balanceada** si exactamente la mitad del dominio va a 0 y la otra mitad va a 1.

Problema: Dada una función $f : \{0,1\}^n \rightarrow \{0,1\}$ balanceada o constante, la cual no podemos observar su definición. Queremos determinar si esta función es constante o balanceada.

Solución clásica: Vamos a tener que evaluar la función f tantas veces como sea necesario. El mejor caso es tras 2 ejecuciones, ya que puede darse la situación en la que es balanceada y cada ejecución da un resultado diferente. El peor caso se da con $2^{n-1} + 1$ ejecuciones, ya que si las primeras 2^{n-1} son iguales, la siguiente nos va a determinar si es constante o balanceada.

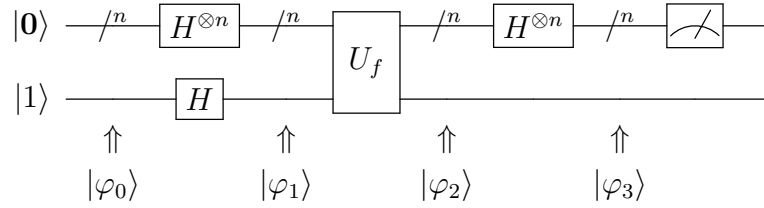
²<https://github.com/sinugarc/TFG/blob/main/Deutsch.ipynb>

Vamos a partir del circuito inicial, donde U_f determina la función f :



Hay que destacar que $|\mathbf{x}\rangle = |x_0x_1\dots x_{n-1}\rangle$, por eso posteriormente escribimos $/^n$, para indicar que son n qubits.

Algoritmo de Deutsch-Jozsa:



Que en término de matrices es: $(H^{\otimes n} \otimes I) U_f (H^{\otimes n} \otimes H) |0, 1\rangle$

Antes de analizar los estados en cada $|\varphi_i\rangle$, vamos a estudiar que ocurre cuando tenemos la puerta $H^{\otimes n}$ y como se comporta sobre los diferentes estados, así como la notación que utilizamos, ya que esta será necesaria en el análisis de los $|\varphi_i\rangle$.

El estudio sobre las características de $H^{\otimes n}$ se pueden estudiar en profundidad en *Quantum computing for computer scientists* [1], aquí resumiremos los resultados principales que nos van a permitir profundizar en los circuitos cuánticos, ya que la aplicación de esta puerta es muy común.

Antes de meternos en $H^{\otimes n}$, vamos a definir un producto interno:

$$\langle , \rangle : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$$

Si $\mathbf{x} = x_0x_1\dots x_{n-1}$ e $\mathbf{y} = y_0y_1\dots y_{n-1}$, entonces:

$$\begin{aligned} \langle \mathbf{x}, \mathbf{y} \rangle &= \langle x_0x_1\dots x_{n-1}, y_0y_1\dots y_{n-1} \rangle \\ &= (x_0 \wedge y_0) \oplus (x_1 \wedge y_1) \oplus \dots \oplus (x_{n-1} \wedge y_{n-1}) \\ &= x_0y_0 \oplus x_1y_1 \oplus \dots \oplus x_{n-1}y_{n-1} \end{aligned} \tag{3.5}$$

Donde \wedge es la operación lógica AND y \oplus la operación lógica XOR. También definimos, con la misma notación, $\mathbf{x} \oplus \mathbf{y} = x_0 \oplus y_0, x_1 \oplus y_1, \dots, x_{n-1} \oplus y_{n-1}$ que tal y como se estudió en la Sección 3.1 es la suma binaria bit a bit.

Con esto obtenemos las siguiente propiedades del producto interno:

- $\langle \mathbf{x} \oplus \mathbf{x}', \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle \oplus \langle \mathbf{x}', \mathbf{y} \rangle$
- $\langle \mathbf{x}, \mathbf{y} \oplus \mathbf{y}' \rangle = \langle \mathbf{x}, \mathbf{y} \rangle \oplus \langle \mathbf{x}, \mathbf{y}' \rangle$
- $\langle 0 \cdot \mathbf{x}, \mathbf{y} \rangle = \langle 0, \mathbf{y} \rangle = 0$
- $\langle \mathbf{x}, 0 \cdot \mathbf{y} \rangle = \langle \mathbf{x}, 0 \rangle = 0$

Una vez definidas estas operaciones y propiedades vamos a poder definir la matriz que caracteriza a $H^{\otimes n}$. Cabe recordar que cada fila y cada columna representa un elemento de la base, que se corresponde con la representación binaria de la fila y la columna desde 0:

$$H^{\otimes n}[\mathbf{i}, \mathbf{j}] = \frac{1}{\sqrt{2^n}} (-1)^{\langle \mathbf{i}, \mathbf{j} \rangle} \quad (3.6)$$

Desde aquí podemos obtener las 2 expresiones que utilizaremos a continuación:

$$H^{\otimes n}|\mathbf{0}\rangle = H^{\otimes n}[-, \mathbf{0}] = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x} \in \{0,1\}^n} |\mathbf{x}\rangle \quad (3.7)$$

$$H^{\otimes n}|\mathbf{y}\rangle = H^{\otimes n}[-, \mathbf{y}] = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{\langle \mathbf{x}, \mathbf{y} \rangle} |\mathbf{x}\rangle \quad (3.8)$$

Ahora ya sí que vamos a observar los estados que se producen en los distintos momentos del circuito:

- $|\varphi_0\rangle = |\mathbf{0}, 1\rangle$
- $|\varphi_1\rangle = \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} |\mathbf{x}\rangle}{\sqrt{2^n}} \right] \left[\frac{|\mathbf{0}\rangle - |\mathbf{1}\rangle}{\sqrt{2}} \right]$
- $|\varphi_2\rangle = \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} |\mathbf{x}\rangle}{\sqrt{2^n}} \right] \left[\frac{|\mathbf{0}\rangle - |\mathbf{1}\rangle}{\sqrt{2}} \right]$

- Ahora realizamos la superposición, $H^{\otimes n}$, sobre una superposición de estados $|\mathbf{x}\rangle$ distintos. Así finalizamos:

$$\begin{aligned}
|\varphi_3\rangle &= \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} \sum_{\mathbf{z} \in \{0,1\}^n} (-1)^{\langle \mathbf{z}, \mathbf{x} \rangle} |\mathbf{z}\rangle}{2^n} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \\
&= \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} \sum_{\mathbf{z} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} (-1)^{\langle \mathbf{z}, \mathbf{x} \rangle} |\mathbf{z}\rangle}{2^n} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \\
&= \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} \sum_{\mathbf{z} \in \{0,1\}^n} (-1)^{f(\mathbf{x}) \oplus \langle \mathbf{z}, \mathbf{x} \rangle} |\mathbf{z}\rangle}{2^n} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]
\end{aligned} \tag{3.9}$$

El último paso a analizar y que nos determinará la validez o el que esperamos de este algoritmo es al medición de los primeros n -qubits. Otra manera de estudiar la medición es directamente preguntarnos cual es la probabilidad de que $|\varphi_3\rangle$ colapse a $|\mathbf{0}\rangle$. Esto es análogo a $|\mathbf{z}\rangle = |\mathbf{0}\rangle$, que por las propiedades estudiadas anteriormente $\langle \mathbf{z}, \mathbf{x} \rangle = \langle \mathbf{0}, \mathbf{x} \rangle = 0$. Con este resultado, si volvemos a la ecuación 3.9:

$$|\varphi_3\rangle = \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} |\mathbf{0}\rangle}{2^n} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \tag{3.10}$$

Veamos cual sería el estado del primer qubit dependiendo de nuestras posibilidades:

- f es constante: $\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} |\mathbf{0}\rangle}{2^n} = \frac{\sum_{\mathbf{x} \in \{0,1\}^n} (\pm 1) |\mathbf{0}\rangle}{2^n} = \frac{(\pm 1) 2^n |\mathbf{0}\rangle}{2^n} = (\pm 1) |\mathbf{0}\rangle$
- f es balanceada: $\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} |\mathbf{0}\rangle}{2^n} = \frac{0 |\mathbf{0}\rangle}{2^n} = 0 |\mathbf{0}\rangle$

Por lo que podemos concluir que este algoritmo resuelve nuestro problema con una única evaluación de f . Si el resultado es $|\mathbf{0}\rangle$, f es constante y si es otro cualquiera, f es balanceada.

Observación: Hay que tener en cuenta, que si la función no es ni balanceada ni constante, este algoritmo no soluciona el problema y el resultado no es concluyente. Esto se debe a que cualquier valor que nos devuelva el algoritmo tiene un interpretación, $|\mathbf{0}\rangle$ si f constante y si no, f es balanceada. Es decir, si f no es ni constante ni balanceada, el resultado sea el que sea va a ser erróneo. Y esto explica porque es una de nuestras hipótesis.

Oráculo: El oráculo que utilizamos para este algoritmo puede venir de dos sitios distintos. Podemos programarlo con ayuda de qiskit ³, ya que nos explica como crear el oráculo de DJ, sea la función balanceada o constante. O bien, una vez definida la función crear la matriz que representa al operador.

Simulaciones: En el repositorio ⁴ podemos encontrar este algoritmo programado con ayuda de *Qiskit*. Una de las peculiaridades que encontramos es la forma en la que *Qiskit* nos presenta los qubits, más bien hay una permutación en la base respecto a la generalmente utilizamos al formalizar matrices. Esto puede ser un problema cuando las puertas las creamos como un operador a partir de la matriz que las determina. El ejemplo de esta situación está programado, así como pruebas del algoritmo de DJ con la misma matriz.

Si bien es cierto que todas las ejecuciones del algoritmo que tiene programado un oráculo balanceado, siguiendo el procedimiento de Qiskit, obtienen como resultado el ket $|1\rangle$, esto no tiene por qué ser cierto en general. Si recordamos el análisis final del algoritmo, si obtenemos $|0\rangle$ entonces f es constante y si no, f es balanceada. Es decir, deberíamos de ser capaces de obtener un resultado distinto de $|0\rangle$ y $|1\rangle$. Esto lo podemos observar en el algoritmo programado en el repositorio con un operador matricial. El resultado obtenido no es ni $|0\rangle$, ni $|1\rangle$, sino que resulta ser $|01\rangle$.

3.4. Bernstein-Vazirani

El algoritmo de Bernstein-Vazirani, fue presentado por Ethan Bernstein y Umesh Vazirani [14]. Se puede entender que este algoritmo es una generalización más del algoritmo de Deutsch-Jozsa⁵.

Problema: Sean $f : \{0, 1\}^n \rightarrow \{0, 1\}$ y $\mathbf{s} = s_0 s_1 \dots s_{n-1} \in \{0, 1\}^n$, sabemos por hipótesis que $f(\mathbf{x}) = \mathbf{s} \cdot \mathbf{x} \bmod 2 = s_0 x_0 \oplus s_1 x_1 \oplus \dots \oplus s_{n-1} x_{n-1}$. Queremos obtener la cadena \mathbf{s} .

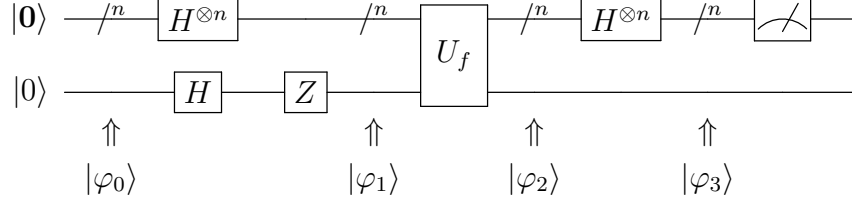
Solución clásica: Ejecutamos la función f n veces una con cada elemento de la base canónica. Esto nos permitirá obtener los n elementos de \mathbf{s} .

³<https://learn.qiskit.org/course/ch-algorithms/deutsch-jozsa-algorithm>

⁴https://github.com/sinugarc/TFG/blob/main/Deutsch_Jozsa.ipynb

⁵<https://learn.qiskit.org/course/ch-algorithms/bernstein-vazirani-algorithm>

Algoritmo de Bernstein-Vazirani, análogo a Deutsch-Jozsa



De forma matricial: $(H^{\otimes n} \otimes I) U_f (I \otimes Z) (H^{\otimes n} \otimes H) |\mathbf{0}, 0\rangle = (H^{\otimes n} \otimes I) U_f (H^{\otimes n} \otimes H) |\mathbf{0}, 1\rangle$

Veamos ahora los estados alcanzados, similares al circuito anterior:

- $|\varphi_0\rangle = |\mathbf{0}, 0\rangle$
- $|\varphi_1\rangle = \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} |\mathbf{x}\rangle}{\sqrt{2^n}} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]$
- Aplicamos la función f :

$$|\varphi_2\rangle = \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} |\mathbf{x}\rangle}{\sqrt{2^n}} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \quad (3.11)$$

- Ahora tenemos que superponer la superposición, aplicada sobre otros estados $|\mathbf{z}\rangle$:

$$\begin{aligned} |\varphi_3\rangle &= \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} \sum_{\mathbf{z} \in \{0,1\}^n} (-1)^{\langle \mathbf{z}, \mathbf{x} \rangle} |\mathbf{z}\rangle}{2^n} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \\ &= \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} \sum_{\mathbf{z} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} (-1)^{\langle \mathbf{z}, \mathbf{x} \rangle} |\mathbf{z}\rangle}{2^n} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \\ &= \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} \sum_{\mathbf{z} \in \{0,1\}^n} (-1)^{f(\mathbf{x}) \oplus \langle \mathbf{z}, \mathbf{x} \rangle} |\mathbf{z}\rangle}{2^n} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \\ &= \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} \sum_{\mathbf{z} \in \{0,1\}^n} (-1)^{\langle \mathbf{s}, \mathbf{x} \rangle \oplus \langle \mathbf{z}, \mathbf{x} \rangle} |\mathbf{z}\rangle}{2^n} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \\ &= \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} \sum_{\mathbf{z} \in \{0,1\}^n} (-1)^{\langle \mathbf{s} \oplus \mathbf{z}, \mathbf{x} \rangle} |\mathbf{z}\rangle}{2^n} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \end{aligned} \quad (3.12)$$

Tenemos que observar lo que ocurre en ese primer bloque de qubits, que será al que apliquemos la medición, si bien en el algoritmo anterior nos interesaba medir cuando $|\mathbf{z}\rangle = |0\rangle$, ahora queremos $|\mathbf{z}\rangle = |\mathbf{s}\rangle$. Es decir, queremos analizar la amplitud del estado $|\mathbf{s}\rangle$.

$$\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{\langle \mathbf{s} \oplus \mathbf{s}, \mathbf{x} \rangle} |\mathbf{s}\rangle}{2^n} = \frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{\langle 0, \mathbf{x} \rangle} |\mathbf{s}\rangle}{2^n} = \frac{\sum_{\mathbf{x} \in \{0,1\}^n} |\mathbf{s}\rangle}{2^n} = |\mathbf{s}\rangle \quad (3.13)$$

¿Qué está ocurriendo aquí?, lo que hemos observado es que la amplitud de $|\mathbf{s}\rangle$ es 1, y si recordamos la teoría de los sistemas cuánticos, la suma de los cuadrados de las amplitudes es 1. Es decir, las amplitudes del resto de posibles valores de $|\mathbf{z}\rangle$ es 0. Por lo que al hacer la medición sobre el primer bloque de qubits nos devolverá la cadena \mathbf{s} deseada, al ser su probabilidad de aparición 1. Podemos concluir entonces que hemos obtenido la cadena \mathbf{s} con una única ejecución de f , mejorando así la solución clásica para este algoritmo.

Oráculo: Antes de presentar las simulaciones realizadas, vamos a estudiar como a partir de una cadena \mathbf{s} podemos construir un oráculo funcione como f_s . Según el valor en la cadena se ejecuta una puerta y otra. Primero invertiremos los valores de \mathbf{s} y a continuación, si el valor es 0, entonces pondremos una puerta identidad, aunque no es necesaria, en la línea del qubit de esa posición y si el valor es 1, pondremos una puerta CNOT entre el qubit de la posición y el qubit ancilla, donde el qubit de control es el de la posición del valor. Los generará la función $bv_fgenerator(qc, n0, s)$, veamos un par de ejemplos ahora:

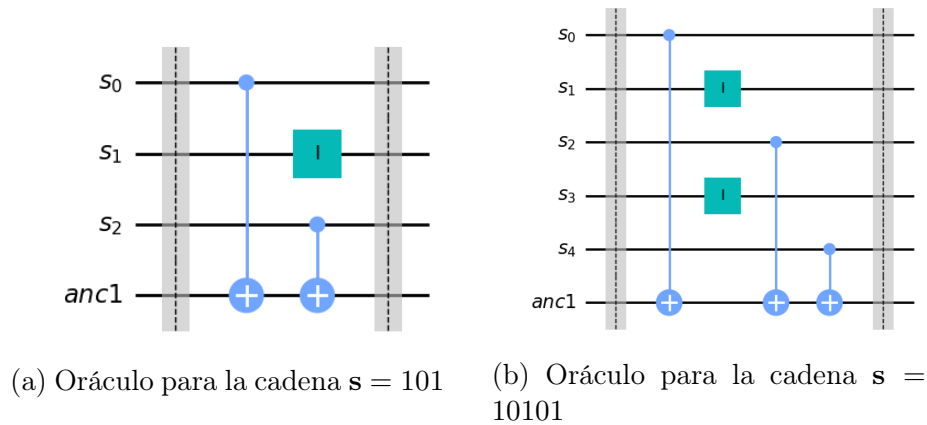


Figura 3.3: Ejemplo de oráculos para el algoritmo de BV

Simulaciones: Las simulaciones de este algoritmo ya han sido presentadas a lo largo del texto, para ampliar estos ejemplos así como revisar el código, se pueden ver en el repositorio⁶.

⁶https://github.com/sinugarc/TFG/blob/main/Bernstein_Vazirani.ipynb

3.5. Simon

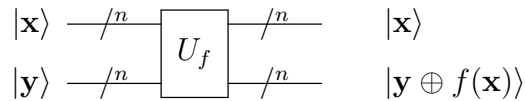
El algoritmo de Simon, o de periodicidad de Simon, va a ser el primero que no va a usar únicamente computación cuántica, sino que la combina con computación clásica al final del mismo. Veremos cual es el problema y la parte cuántica de la solución, para así ver como procedemos posteriormente hasta encontrar la solución buscada.

Sea $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$, la cual no conocemos su definición, pero si sabemos es periódica en el sentido de que existe una cadena binaria $\mathbf{c} = c_0 c_1 \dots c_{n-1}$ tal que $\forall \mathbf{x}, \mathbf{y} \in \{0, 1\}^n$ entonces, $f(\mathbf{x}) = f(\mathbf{y})$ si y solo si $\mathbf{x} = \mathbf{y} \oplus \mathbf{c}$, donde \oplus es la suma binaria bit a bit. Llamaremos \mathbf{c} al **periodo** de f .

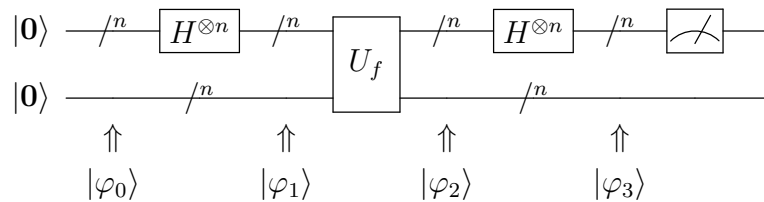
Problema: Dada una función $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ con periodo $\mathbf{c} = c_0 c_1 \dots c_{n-1}$. Queremos determinar cual es el periodo de f teniendo en cuenta que no conocemos su definición.

Solución clásica: Vamos ejecutando f hasta encontrar una coincidencia. En el peor de los casos, la función es inyectiva y por lo tanto $\mathbf{c} = \mathbf{0}$, para esto habrá que ejecutar f al menos sobre la mitad del dominio, es decir, $2^{n-1} + 1$ ejecuciones. Esto se debe a que si existiera un periodo distinto de $\mathbf{0}$, ya habríamos encontrado alguna coincidencia entre la imagen de 2 elementos del dominio.

Vamos a partir del circuito inicial, donde U_f determina la función f :



Algoritmo de Simon, parte cuántica:



Si lo observamos como operación matricial: $(H^{\otimes n} \otimes I) U_f (H^{\otimes n} \otimes I) |\mathbf{0}, \mathbf{0}\rangle$

Análogamente al algoritmo de Deutsch-Jozsa, analizamos los estados del circuito:

- $|\varphi_0\rangle = |\mathbf{0}, \mathbf{0}\rangle$
- $|\varphi_1\rangle = \frac{\sum_{\mathbf{x} \in \{0,1\}^n} |\mathbf{x}, \mathbf{0}\rangle}{\sqrt{2^n}}$
- $|\varphi_2\rangle = \frac{\sum_{\mathbf{x} \in \{0,1\}^n} |\mathbf{x}, f(\mathbf{x})\rangle}{\sqrt{2^n}}$
- $|\varphi_3\rangle = \frac{\sum_{\mathbf{x} \in \{0,1\}^n} \sum_{\mathbf{z} \in \{0,1\}^n} (-1)^{\langle \mathbf{z}, \mathbf{x} \rangle} |\mathbf{z}, f(\mathbf{x})\rangle}{2^n}$

Si bien antes era algo más directo que nos proporcionaba el resultado, aquí tenemos que analizarlo un poco más, ver hasta donde llega nuestro algoritmo cuántico y que es lo que obtenemos de él. La propiedad principal es la periodicidad de f , por lo que por definición $|\mathbf{z}, f(\mathbf{x})\rangle = |\mathbf{z}, f(\mathbf{x} \oplus \mathbf{c})\rangle$. Vamos a estudiar el coeficiente al agruparlos, al ser mismo ket:

$$\begin{aligned} \frac{(-1)^{\langle \mathbf{z}, \mathbf{x} \rangle} + (-1)^{\langle \mathbf{z}, \mathbf{x} \oplus \mathbf{c} \rangle}}{2} &= \frac{(-1)^{\langle \mathbf{z}, \mathbf{x} \rangle} + (-1)^{\langle \mathbf{z}, \mathbf{x} \rangle \oplus \langle \mathbf{z}, \mathbf{c} \rangle}}{2} \\ &= \frac{(-1)^{\langle \mathbf{z}, \mathbf{x} \rangle} + (-1)^{\langle \mathbf{z}, \mathbf{x} \rangle} (-1)^{\langle \mathbf{z}, \mathbf{c} \rangle}}{2} \end{aligned} \quad (3.14)$$

Por lo que si $\langle \mathbf{z}, \mathbf{c} \rangle = 1$, entonces los sumandos se cancelan. Por otra parte, si $\langle \mathbf{z}, \mathbf{c} \rangle = 0$ el coeficiente será (± 1) , es decir, al realizar la medición sobre el primer bloque de qubits, solo obtendremos aquellos tales que $\langle \mathbf{z}, \mathbf{c} \rangle = 0$.

Pero, ¿proporciona esto una solución esto nuestro algoritmo? Visto así directamente puede parecer que no, debido a que cada vez que lo ejecutemos nos proporcionará una cadena distinta que tiene esa característica. Pero tras obtener n cadenas, obtenemos un sistema de ecuaciones a partir del cual obtenemos la cadena \mathbf{c} deseada. Aquí es donde entraría la computación clásica para acabar de resolver el problema.

Se puede ver la mejora respecto a la solución clásica, ya que al ser todas las cadenas equiprobables, obtendremos n cadenas distintas en menos ejecuciones que $2^{n-1} + 1$, para n suficientemente grande. Se puede ver un ejemplo concreto del desarrollo completo del algoritmo con $f : \{0,1\}^3 \rightarrow \{0,1\}^3$ en *Quantum computing for computer scientists*, 190 [1].

Oráculo: Se importa directamente desde Qiskit.

Simulaciones: Debido a que utilizamos este circuito, sin ninguna modificación, al comprobar nuestras MR, las simulaciones se encuentran en el repositorio común, en el archivo sobre las reglas del algoritmo de Simon⁷.

El circuito, análogo al teórico, para el algoritmo de Simon con una cadena \mathbf{c} de longitud 3 es:

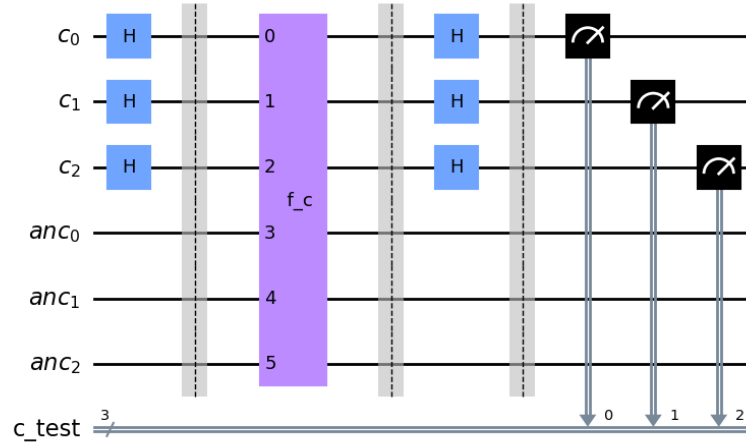


Figura 3.4: Circuito, algoritmo Simon con $\mathbf{c} = 100$

Lo hemos simulado y ejecutado en un sistema cuántico de IBM y los resultados son equivalentes. Además de observar el ruido del sistema cuántico, se va a poder apreciar ambas formas de realizar el histograma, ya sea con conteo o probabilidades.

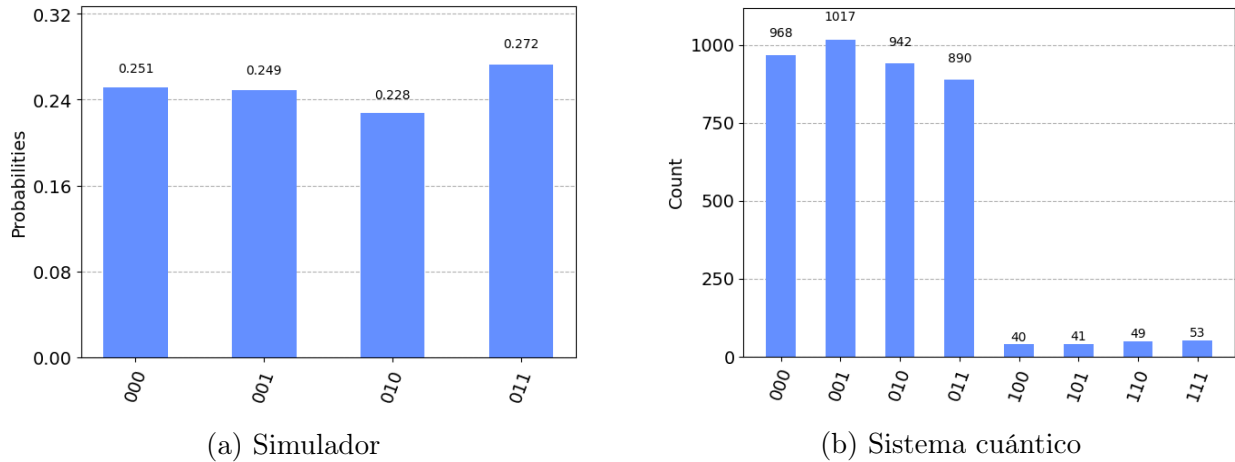


Figura 3.5: Resultados del algoritmo Simon para la cadena $\mathbf{c} = 100$.

⁷https://github.com/rodelanu/TFG/blob/main/3_Simon_Rules.ipynb

Capítulo 4

Propiedades metamórficas

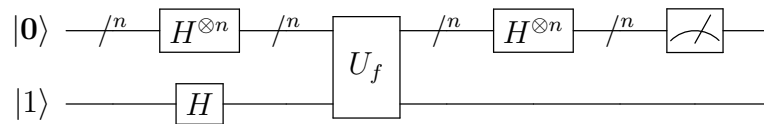
En este último capítulo antes de la conclusión queremos presentar al lector con el objetivo principal de este trabajo. Queremos finalmente unir las propiedades metamórficas con los algoritmos del capítulo 3, de los cuales se estudiarán únicamente los tres últimos. A continuación, presentaremos cuales son las propiedades obtenidas, así como de donde proceden, para luego pasar a la programación y los circuitos que hemos preparado para poder realizar las pruebas necesarias.

4.1. Deutsch-Jozsa

En esta sección vamos a estudiar qué reglas hemos obtenido del algoritmo de Deutsch-Jozsa y cómo las hemos implementado en nuestro repositorio. Recordamos brevemente el problema y el algoritmo obtenido:

Problema: Dada una función $f : \{0, 1\}^n \rightarrow \{0, 1\}$ balanceada o constante, la cual no podemos observar su definición. Queremos determinar si esta función es constante o balanceada.

Algoritmo de Deutsch-Jozsa:



Resultado: Se obtiene $|0\rangle$ si f es constante y cualquier otro resultado si f es balanceada.

Observación: No hay que olvidar la importancia de tener como hipótesis que f es constante o balanceada. Pero en este caso, debido a que esta va a formar parte de nuestro *source input*, podemos asegurar esta propiedad.

Suponemos P la implementación del algoritmo de Deutsch-Jozsa a analizar, veamos qué reglas hemos obtenido y su implementación:

Regla I: Sea $f : \{0, 1\}^n \rightarrow \{0, 1\}$ la función a analizar definida en problema y sea $g : \{0, 1\}^n \rightarrow \{0, 1\}^n$ un automorfismo. Entonces, $P(f) = |\mathbf{0}\rangle$ si y solo si $P(f \circ g) = |\mathbf{0}\rangle$.

Al aplicar el automorfismo g , $f \circ g$ no varía el número de 1 y 0 de la imagen y por lo tanto no cambia la característica de la función, por lo que es directo decir que la MR es correcta.

Implementación: Comparamos los resultados entre la simulación $P(f)$ y $P(f \circ g)$. En este caso hemos decidido utilizar un automorfismo particular a la hora de realizar las pruebas. Sea $g : \{0, 1\}^n \rightarrow \{0, 1\}^n$ talque aplica la puerta X a cada componente, es decir, $g(\mathbf{x}) = \mathbf{1} - \mathbf{x}$ que es el complementario binario de \mathbf{x} .

Regla II: Sea $f : \{0, 1\}^n \rightarrow \{0, 1\}$ la función a analizar definida en problema y sea $h : \{0, 1\}^n \rightarrow \{0, 1\}$ tal que $h(x) = 1 - f(x)$. Entonces, $P(f) = |\mathbf{0}\rangle$ si y solo si $P(h) = |\mathbf{0}\rangle$.

Al igual que con la regla I, no se varía la condición de f ya que si esta es balanceada mantiene el número de 0 y 1, ya que tienen el mismo. Y si fuera constante, simplemente cambiaría el valor de la constante.

Implementación: Este caso es análogo al anterior, simplemente cambiamos la condición del qubit ancilla: le aplicamos una puerta X tras el oráculo.

Es cierto que en el repositorio¹, así como en la programación realizada de acuerdo a MT, consideramos que el resultado de P es un valor binario, esto lo obtenemos haciendo el máximo de los bit obtenidos. Cabe recordar que si una función es balanceada el resultado solo implica que no es $|\mathbf{0}\rangle$, es decir en nuestros bits tendremos al menos un uno, que nos determinará el máximo a 1. Por lo que podríamos concluir que a nivel de programación en MT $P \rightarrow 0$ si es constante y $P \rightarrow 1$ si es balanceada. Las reglas expresadas de forma matemáticamente exacta de acorde en al algoritmo en este documento son equivalentes a la expresadas en el repositorio, tras la transformación explicada en este mismo párrafo.

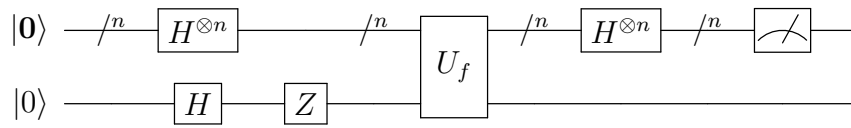
¹https://github.com/rodelanu/TFG/blob/main/1_Deutsch_Jozsa_Rules.ipynb

4.2. Bernstein-Vazirani

El algoritmo de Bernstein-Vazirani fue el primero presentado como ejemplo al introducir la programación cuántica en el capítulo 2. Recapitulemos sobre lo estudiado del algoritmo de BV.

Problema: Sea $f : \{0, 1\}^n \rightarrow \{0, 1\}$, sea $\mathbf{s} = s_0 s_1 \dots s_{n-1} \in \{0, 1\}^n$, por hipótesis sabemos que $f(\mathbf{x}) = (\mathbf{s} \cdot \mathbf{x}) \bmod 2 = s_0 x_0 \oplus s_1 x_1 \oplus \dots \oplus s_{n-1} x_{n-1}$. Queremos obtener la cadena \mathbf{s} .

Algoritmo de Bernstein-Vazirani:



Resultado: Obtenemos la cadena $|\mathbf{s}\rangle$ en la medición.

Si suponemos que P es una implementación del algoritmo de BV, vamos a estudiar la reglas obtenidas:

Regla I: Sea \mathbf{s} una cadena binaria de longitud n y sea $\bar{\mathbf{s}}$ su complementario binario. Entonces $P(\mathbf{s}) \oplus P(\bar{\mathbf{s}}) = |\mathbf{1}\rangle$ de longitud n .

Esta regla se ha obtenido como un caso particular de la regla general que se presentará a continuación. Ahora bien, ¿Es trascendente el particularizar una regla ya obtenida? Hagamos un pequeño inciso para aclarar este tema.

La respuesta a esta pregunta es sí, ya que aunque una regla más general cubre muchos más casos, esto no nos asegura que encontrar un fallo sea más fácil. Sin embargo, puede ser incluso más complejo programar un regla general. Es cierto que este caso es muy simple y la comprobación para es directa, ya que queremos obtener el ket $|\mathbf{1}\rangle$.

El ejemplo básico que se usa para explicar esta diferencia es el siguiente [5]:

Suponemos que tenemos una implementación P con una MR tal que $f(k \times x) = k \times f(x)$, donde $k \in \mathbb{Z} \setminus \{0\}$. De esta manera se puede extrapolar fácilmente de un *source input* a infinitos casos al fijar un x y variar $k \in \mathbb{N}$. Pero en realidad, sigue dejando muchísimos casos sin comprobar y tiene cierta dificultad para confirmar todos los *follow-up inputs* en P , debido a la cantidad de comprobaciones a realizar para un mismo *source input*. Por otra parte, si tomamos $k = -1$ tenemos otra MR, que aún siendo más débil que la anterior es más fácil de verificar. Y evidentemente, cualquier fallo que se dé con $f(-x) = -f(x)$ es tan efectivo como uno encontrado con la regla general.

Ya nos podemos centrar en nuestra regla I. Para programar este test, vamos a ejecutar por separado P para cada cadena s y \bar{s} y realizaremos la suma bit a bit de los resultados antes de realizar la medición. La razón por la que esperamos el ket $|1\rangle$ se debe a que si uno es el complementario binario del otro, su suma binaria es 1 en cada posición.

El circuito utilizado es idéntico a la Figura 4.1, simplemente $s1$ cumplirá que $s \oplus s1 = |1\rangle$.

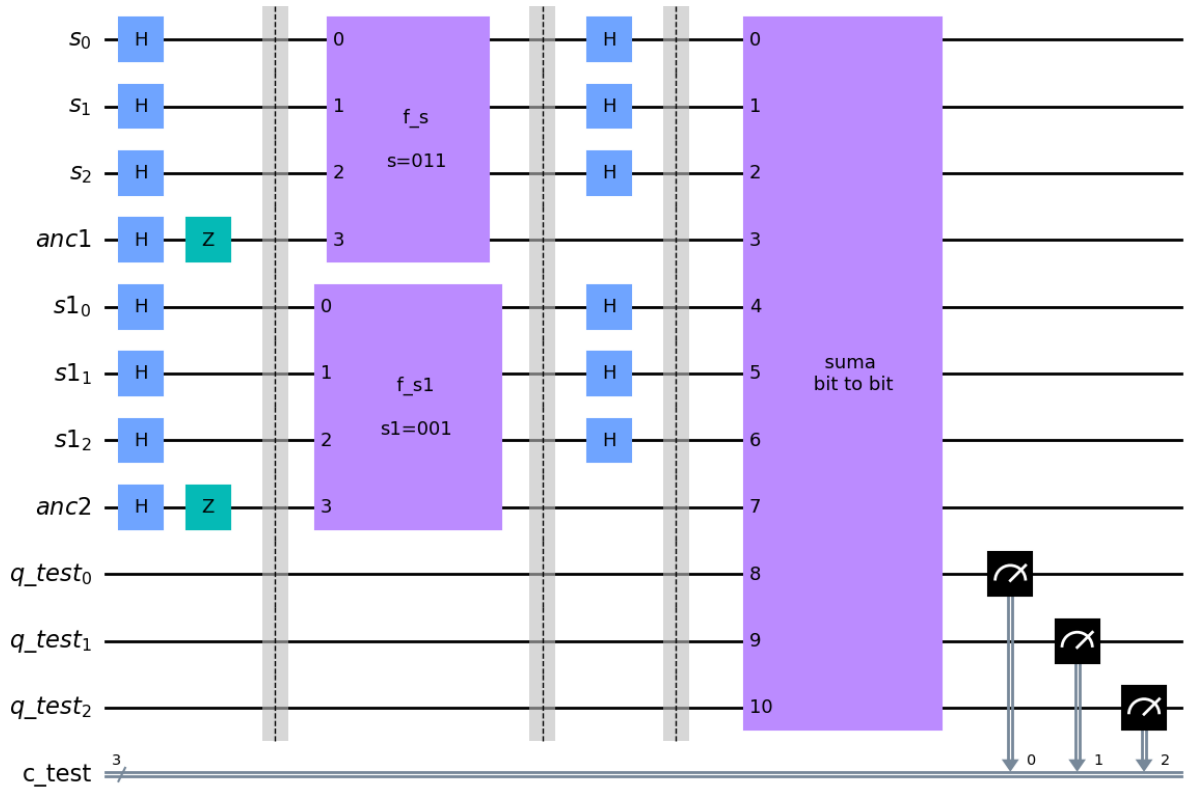


Figura 4.1: Circuito para la regla II de BV

Regla II: Sean \mathbf{s} y \mathbf{s}' cadenas binarias de longitud n . Entonces $P(\mathbf{s}) \oplus P(\mathbf{s}') = \mathbf{s} \oplus \mathbf{s}'$.

Se puede entender que esta es la generalización de la regla I, si bien es cierto podría incluso generalizarse para 2 cadenas que no tengan que tener la misma longitud o incluso se podría interpretar como $P(f_{\mathbf{s}} \oplus f_{\mathbf{s}'}) = P(f_{\mathbf{s}}) \oplus P(f_{\mathbf{s}'})$, relacionando así dos *output* del *source input* con uno del *follow-up output*.

Las reglas anteriores han sido obtenidas de forma directa por la naturaleza del problema, donde si el resultado es la cadena con la que hemos generado el oráculo, entonces la suma binaria se debe conservar tanto en el *source input* como en el *output*.

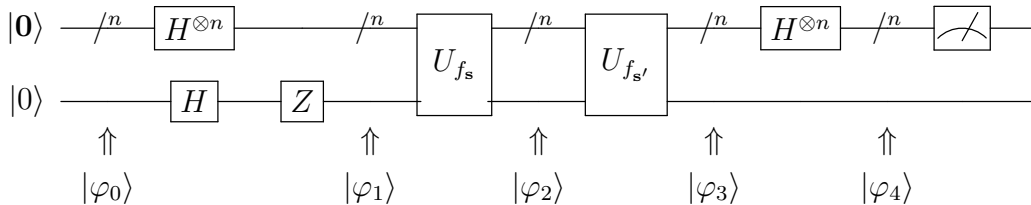
El circuito utilizado se muestra en la Figura 4.1. Es el circuito creado para las cadenas $\mathbf{s} = 011$ y $\mathbf{s}' = 001$, las cuales se generan de manera aleatoria una vez determinada una longitud n que se pasa como parámetro a la función.

Regla III: Sean \mathbf{s} y \mathbf{s}' cadenas binarias de longitud n . Si entendemos la composición como la concatenación de sus oráculos, entonces $P(f(\mathbf{s}) \circ f(\mathbf{s}')) = P(f_{\mathbf{s}}) \oplus P(f_{\mathbf{s}'})$.

La primera vez que se pensó en esta regla fue directamente desde las ecuaciones del algoritmo, en particular, la ecuación 3.11. Tras aplicar la primera f , en este caso $f_{\mathbf{s}}$, obtenríamos:

$$|\varphi_2\rangle = \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f_{\mathbf{s}}(\mathbf{x})} |\mathbf{x}\rangle}{\sqrt{2^n}} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \quad (4.1)$$

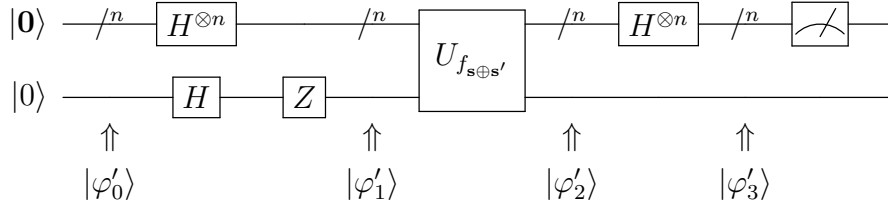
Veamos como quedaría el circuito con la concatenación, que incluye este estado:



Desde aquí podemos desarrollar $|\varphi_3\rangle$:

$$\begin{aligned}
|\varphi_3\rangle &= \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f_{\mathbf{s}}(\mathbf{x})} (-1)^{f_{\mathbf{s}'}(\mathbf{x})} |\mathbf{x}\rangle}{\sqrt{2^n}} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \\
&= \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f_{\mathbf{s}}(\mathbf{x}) \oplus f_{\mathbf{s}'}(\mathbf{x})} |\mathbf{x}\rangle}{\sqrt{2^n}} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \\
&= \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{\langle \mathbf{s}, \mathbf{x} \rangle \oplus \langle \mathbf{s}', \mathbf{x} \rangle} |\mathbf{x}\rangle}{\sqrt{2^n}} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \\
&= \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{\langle \mathbf{s} \oplus \mathbf{s}', \mathbf{x} \rangle} |\mathbf{x}\rangle}{\sqrt{2^n}} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] \\
&= \left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f_{\mathbf{s} \oplus \mathbf{s}'}(\mathbf{x})} |\mathbf{x}\rangle}{\sqrt{2^n}} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]
\end{aligned} \tag{4.2}$$

Con este desarrollo obtenemos directamente la regla III, ya que llegamos exactamente a la misma ecuación desde el circuito definido para $f_{\mathbf{s} \oplus \mathbf{s}'}$, donde $|\varphi_3\rangle = |\varphi'_2\rangle$.



Para la implementación de esta regla, hemos querido reducir la complejidad de las comparaciones y así evitar un uso excesivo de qubits. Hay que recordar que las matrices que usan los simuladores crecen de manera exponencial, ya que la base de n qubits tiene 2^n elementos. Además en IBM, el máximo sistema cuántico de acceso libre tiene sólo 7 qubits, por lo que quizás para esta regla no tuviéramos problemas, pero para la regla I y II, seguramente no podríamos ejecutarlos.

Por lo que implementamos $P(f(\mathbf{s}) \circ f(\mathbf{s}'))$, ya que la suma por separado sería análoga a las reglas anteriores. De esta manera el circuito que utilizaremos a la hora de realizar MT será,

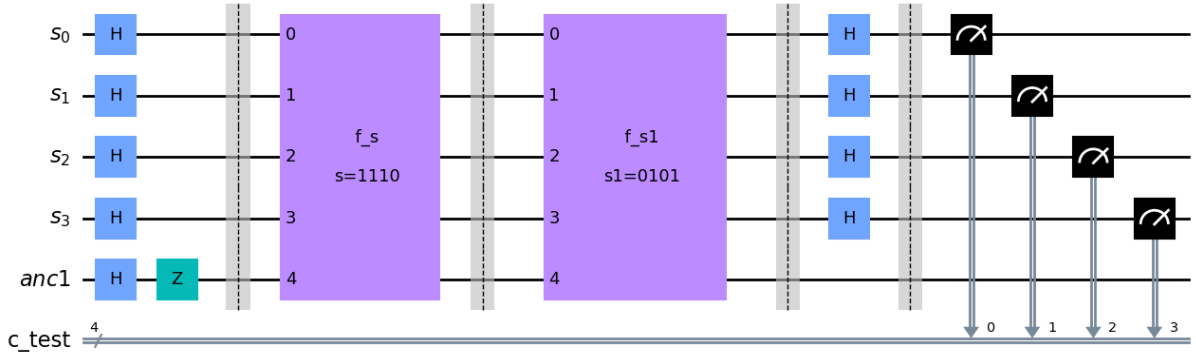


Figura 4.2: Circuito para la regla III de BV

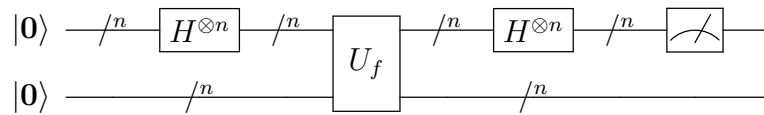
Los circuitos y pruebas realizadas se puede encontrar en el archivo de BV ² del repositorio común.

4.3. Simon

Por último vamos a terminar de estudiar el algoritmo de Simon y como vamos a obtener las reglas metamórficas. En este caso no van a ser una aplicación tan directa como los algoritmos anteriores, debido a que el algoritmo de Simon acaba con computación clásica. Por lo que para la obtención de MR, nos vamos a centrar en ese punto intermedio entre el uso de la computación cuántica y el paso a lo clásico para solucionar los sistemas de ecuaciones que vimos en la presentación del algoritmo en la sección 3.5

Problema: Dada una función $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ con periodo $\mathbf{c} = c_0c_1\dots c_{n-1}$. Queremos determinar cual es el periodo de f teniendo en cuenta que no conocemos su definición.

Algoritmo de Simon, parte cuántica:



Resultados: Obtención de n cadenas binarias tal que si \mathbf{z} es una de estas cadenas, entonces $\langle \mathbf{z}, \mathbf{c} \rangle = 0$.

²https://github.com/rodelanu/TFG/blob/main/2_Bernstein_Vazirani_Rules.ipynb

Supongamos ahora que tenemos P implementación del algoritmo de Simon, veamos que MR hemos podido obtener para este algoritmo.

Regla I: Al efectuar la suma bit a bit con los posibles resultados del programa que resuelve el algoritmo de Simon, obtenemos de nuevo el conjunto inicial. Es decir, forma un grupo con la suma bit a bit.

Vamos a comprobar que es cierto que forma un grupo con esa operación interna, de esta manera ya habremos fundamentado de forma teórica la base de esta regla. Sea S_c el conjunto de las \mathbf{z} cadenas que se puede obtener con la implementación P del algoritmo de Simon para una cierta cadena \mathbf{c} y \oplus la suma bit a bit:

- $S_c \neq \emptyset$: Veamos que $\mathbf{0} \in S_c$, $\forall \mathbf{c} \in \{0, 1\}^n$. Sea $\mathbf{c} \in \{0, 1\}^n$, $\langle \mathbf{0}, \mathbf{c} \rangle = 0 \Rightarrow \mathbf{0} \in S_c$
- \oplus es una operación interna: Sean $\mathbf{c} \in \{0, 1\}^n$ y $\mathbf{z}, \mathbf{z}' \in S_c$. Tenemos que comprobar que $\mathbf{z} \oplus \mathbf{z}' \in S_c$:

$$\langle \mathbf{z} \oplus \mathbf{z}', \mathbf{c} \rangle = \langle \mathbf{z}, \mathbf{c} \rangle \oplus \langle \mathbf{z}', \mathbf{c} \rangle = 0 \oplus 0 = 0 \Rightarrow \mathbf{z} \oplus \mathbf{z}' \in S_c \quad (4.3)$$

- **Elemento neutro:** Veamos que $\mathbf{0}$ es elemento neutro, ya se probó en $S_c \neq \emptyset$ que $\mathbf{0} \in S_c$, $\forall \mathbf{c} \in \{0, 1\}^n$. Dada $\mathbf{z} \in S_c$, $\mathbf{z} \oplus \mathbf{0} = \mathbf{z} = \mathbf{0} \oplus \mathbf{z}$, se verifica $\forall \mathbf{z} \in \{0, 1\}^n$.
- **Elemento inverso:** Queremos ver que si $\mathbf{z} \in S_c$ existe $\mathbf{z}^{-1} \in S_c$ tal que $\mathbf{z} \oplus \mathbf{z}^{-1} = \mathbf{0}$. Por las propiedades de la suma bit a bit, sabemos que \mathbf{z} es su propio inverso, es decir $\mathbf{z}^{-1} = \mathbf{z}$, debido a que para cada posición de la cadena ambos son 0 o 1 y la suma del bit, siempre es 0. Por lo que, si $\mathbf{z} \in S_c \Rightarrow \mathbf{z}^{-1} = \mathbf{z} \in S_c$ tal que $\mathbf{z} \oplus \mathbf{z}^{-1} = \mathbf{z} \oplus \mathbf{z} = \mathbf{0}$.
- **Asociatividad:** Sean $\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3 \in S_c$, $(\mathbf{z}_1 \oplus \mathbf{z}_2) \oplus \mathbf{z}_3 = \mathbf{z}_1 \oplus (\mathbf{z}_2 \oplus \mathbf{z}_3)$ al ser \oplus asociativa entonces se cumple la igualdad anterior.

Por lo que ya hemos probado que $\forall \mathbf{c} \in \{0, 1\}^n$, (S_c, \oplus) es un grupo.

Implementación: Para la implementación de esta regla hemos creado dos circuitos separados, una vez elegido un $\mathbf{c} \in \{0, 1\}^n$, hemos creado el oráculo para esa cadena, nosotros hemos tomado $\mathbf{c} = |001\rangle$ para dibujar los circuitos y $\mathbf{c} = |0001\rangle$ para la obtención de soluciones, ya que queríamos evitar que el circuito fuera demasiado grande a la hora de mostrarlo.

El primer circuito, Figura 3.4, es el mismo que se presentó en el algoritmo de Simon, donde obtenemos la cadenas \mathbf{z} . El segundo circuito, Figura 4.3, lo que hacemos es aplicar en paralelo el algoritmo y sumar bit a bit ambos resultados antes de medir, de esta manera solo nos quedaría comprobar que los resultados son iguales. Que como se podrá observar en la Figura 4.4, son idénticos al realizar ambas simulaciones.

Regla II: Si simulamos la implementación del algoritmo a una cadena \mathbf{c} y se invierten las cadenas $\mathbf{z} \in S_{\mathbf{c}}$, equivale a aplicar el programa a \mathbf{c} invertida.

Esto se debe a que si permutamos los bits para realizar la inversión tanto en el *source input* como en el *output*, no estamos variando el resultado. Se entiende de forma más sencilla con una perspectiva más matemática. Sean M la matriz que realiza la inversión, no hay que olvidar que $M = M^{-1}$, y $\mathbf{z} \in S_{\mathbf{c}}$:

$$\begin{aligned}\langle \mathbf{z}, \mathbf{c} \rangle &= z_1 c_1 \oplus z_2 c_2 \oplus \dots \oplus z_n c_n \\ &= z_n c_n \oplus \dots \oplus z_2 c_2 \oplus z_1 c_1 \\ &= \langle M\mathbf{z}, M\mathbf{c} \rangle = 0 \Rightarrow M\mathbf{z} \in S_{M\mathbf{c}}\end{aligned}\tag{4.4}$$

Esto se puede realizar de esta manera directa debido a que \oplus es un operador conmutativo, por lo que incluso podríamos ampliar a que $(S_{\mathbf{c}}, \oplus)$ es un grupo abeliano. Esto probaría la validez matemática de esta MR para el algoritmo de Simon.

Implementación: simulamos el algoritmo original sobre \mathbf{c} y $M\mathbf{c}$, invirtiendo una de las listas de cadenas obtenidas para comparar resultados. Estos pueden ser representados directamente en forma de listas, tal y como está en el documento o con la representación en histograma tras hacer la modificación.

Todas estas simulaciones, así como el código para ambas reglas se pueden encontrar en el repositorio común, específicamente en el archivo sobre las reglas del algoritmo de Simon³.

Para finalizar, habría que destacar que, tras realizar las pruebas implementadas, los resultado obtenidos han sido los esperados en todas las reglas propuestas en este texto.

³https://github.com/rodelanu/TFG/blob/main/3_Simon_Rules.ipynb

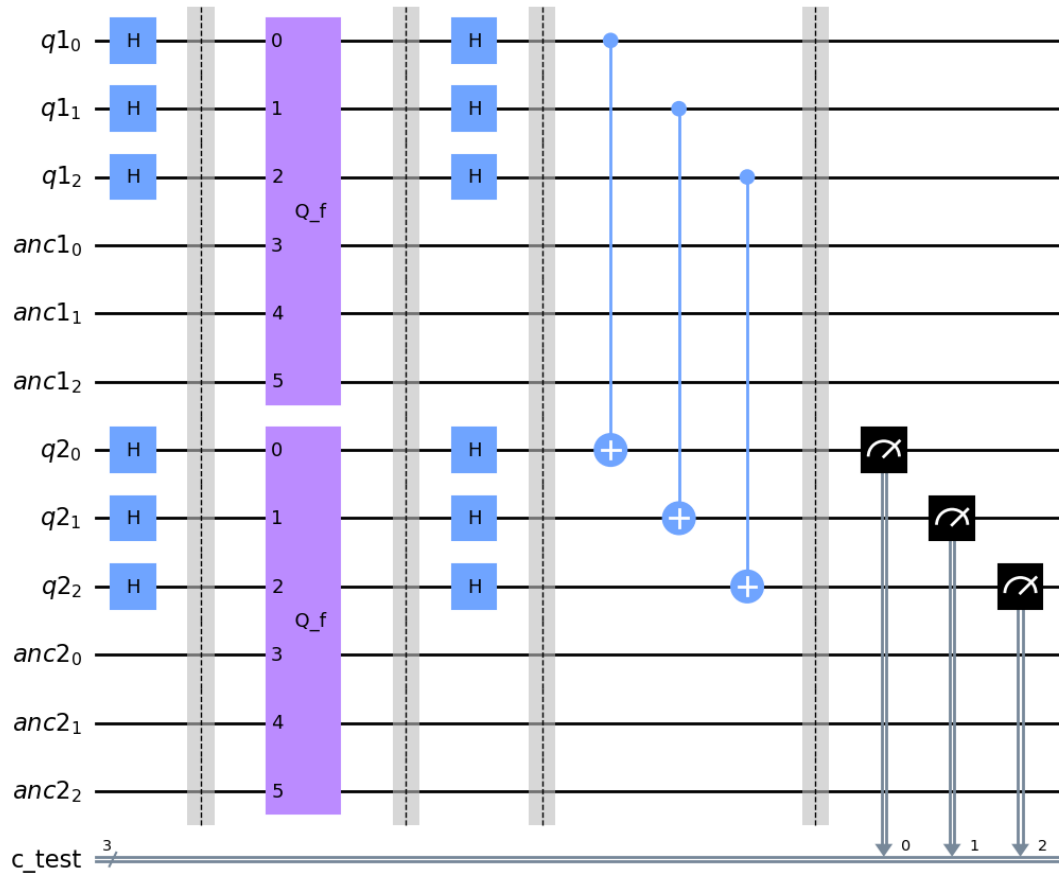
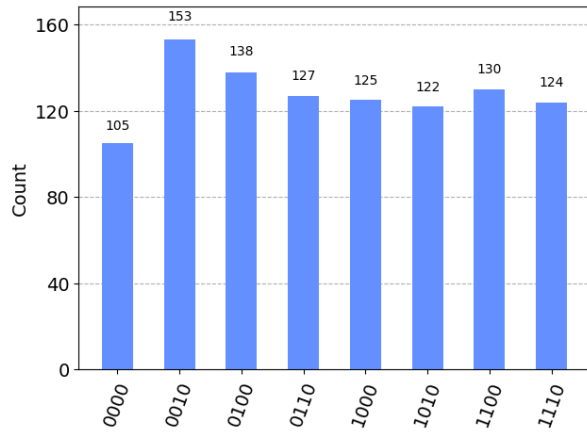
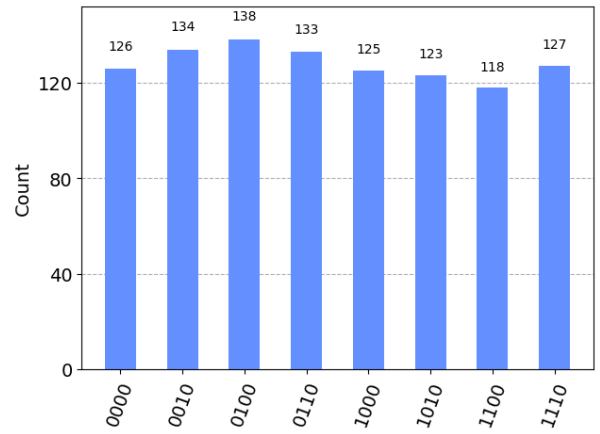


Figura 4.3: Circuito, algoritmo Simon para la regla I con $\mathbf{c} = |001\rangle$



(a) Algoritmo original



(b) Regla I

Figura 4.4: Resultados de la simulación del algoritmo Simon en el circuito original y en el circuito con la regla I implementada, para la cadena $\mathbf{c} = |0001\rangle$.

Capítulo 5

Conclusiones y trabajo futuro

Una vez que se han presentado las MR, así como la aplicación del MT en nuestros algoritmos cuánticos, que es el objetivo principal del trabajo, pasaremos a analizar todo este largo recorrido y los resultados obtenidos. También exploraremos las posibilidades hemos dejado abiertas a lo largo del texto.

5.1. Conclusiones

Lo primero que me gustaría destacar de este trabajo es la cantidad de materia que se ha tenido que revisar y comprender para poder alcanzar nuestro objetivo. Esto ha sido altamente enriquecedor para la comprensión desde un punto de vista más avanzado. Si bien es cierto que todo este aprendizaje ha consumido gran parte del tiempo que se ha dedicado a la realización del TFG, esto se refleja directamente sobre la proporción de páginas que se dedican a esta preparación, que incluyen el capítulo 2 de antecedentes y el capítulo 3 de algoritmos cuánticos.

Desde mi punto de vista, lo más importante e interesante de este proceso de preparación ha sido comprender la relación directa que existe entre los postulados cuánticos y todos los elementos de la programación cuántica. Se puede establecer una correspondencia uno a uno, que he intentado expresar y guiar al lector a través de la introducción a la programación y Qiskit. Además, ha sido importante entender que no todo lo relacionado con lo cuántico es estrictamente probabilístico, sino que tiene sus fundamentos deterministas que se transforman en probabilidades cuando el sistema es observado.

Una vez establecidas estas bases, nos hemos dedicado al estudio las propiedades meta-mórficas y como estas MR pueden ayudarnos a encontrar fallos en nuestros algoritmos. La conservación de estas propiedades intrínsecas del algoritmo nos ayuda con el *testing* de programas cuánticos, independientemente de su complejidad. Además, dado que su estructura está determinada por la definición de MT, su implementación resulta sencilla.

Por último, me gustaría destacar la necesidad y el respaldo que las matemáticas brindan a todas las ciencias. En el caso de la programación cuántica, la base principal para la creación y verificación determinista de los algoritmos es especialmente matemática. De igual manera, la obtención de las MR y la prueba de su validez se fundamentan en principios matemáticos.

Me resultó muy interesante el desarrollo realizado para la creación del algoritmo de Deutsch. Es evidente que los algoritmos no surgen de la nada, sino que se basan en conocimientos más abstractos, como la aplicación de puertas cuánticas y la prueba matemática de su efectividad. En caso de que no lograr el resultado deseado, el análisis realizado nos puede proporcionar ideas para dar el siguiente paso hacia nuestro objetivo. Lo he incluido en este texto para mostrar al lector el camino realizado desde el problema inicial hasta el algoritmo final.

5.2. Trabajo futuro

A lo largo de este texto se han ido dejando puertas abiertas hacia posibilidades de estudio, como las recogidas en los retos del *testing* metamórfico. Estos retos, que intentan guiar a los investigadores hacia nuevas oportunidades o caminos que se deberían completar para una mejor comprensión del MT, se pueden encontrar en el artículo *Metamorphic testing: A new approach for generating next test cases* [5] con un mayor número de retos y profundidad, además este artículo es el que hemos utilizado para el estudio de los conceptos referentes al *testing* metamórfico.

En cuanto a los avances que se han ido realizando en el campo de la computación cuántica y el MT desde que se comenzó este trabajo, podemos destacar el artículo publicado sobre la corrección de las implementaciones de Shor con *testing* metamórfico [15], donde se observa los avances y el potencial que tiene el MT sobre algoritmos más complejos de los presentados en este texto. A su vez, se puede observar la falta de capacidad para realizar todas estas pruebas con los sistemas cuánticos actuales y como se va a necesitar de una mejora en el potencial de estos sistemas para poder llevar este *testing* de manera efectiva sobre algoritmos cuánticos más complejos. Además, este año se ha publicado otro artículo que pone a prueba Qiskit con *testing* metamórfico [16] encontrando fallos en el mismo.

Respecto a la continuación directa sobre el objetivo de este trabajo, se podrían utilizar distintas aproximaciones:

- Profundizar en otros algoritmos como la transformada cuántica de Fourier, en adelante QFT, y sus aplicaciones, como la estimación de fase, QPE, que se comenzaron a estudiar al igual que el algoritmo de Grover. Sin embargo, no tuvimos tiempo para la total comprensión y estudio de reglas metamórficas. Una posible idea para QFT sería utilizar la misma técnica que se usó para obtener la Regla III del algoritmo de BV, ya que el QPE utiliza QFT^{-1} .
- Estudio de otros tipos de *testing* como pruebas de mutación para relacionarlas con MT y poder aplicar ambas a la vez. Esta técnica se utiliza en el artículo *Metamorphic testing of oracle quantum programs* [7].

Para finalizar, solo me faltaría destacar que los algoritmos cuánticos siguen creciendo conforme pasan los días, cada vez aplicados a más ramas de las matemáticas. Una de las mayores colecciones que hemos encontrado es <https://quantumalgorithmzoo.org/>, de donde se podrían obtener otros algoritmos que pudiéramos poner en estudio.

Bibliografía

- [1] Noson S Yanofsky and Mirco A Mannucci. *Quantum computing for computer scientists*. Cambridge University Press, 2008.
- [2] Richard P Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21:467–488, 1982.
- [3] T. Y. Chen, S. C. Cheung, and S. M. Yiu, “Metamorphic testing: A new approach for generating next test cases,” Technical Report HKUST-CS98-01, Department of Computer Science, The Hong Kong University of Science and Technology, Tech. Rep., 1998.
- [4] Michael A Nielsen and Isaac Chuang. *Quantum computation and quantum information*. American Association of Physics Teachers, 2002.
- [5] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, TH Tse, and Zhi Quan Zhou. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys (CSUR)*, 51(1):1–27, 2018.
- [6] Bryan Eastin and Steven T Flammia. Q-circuit tutorial. *arXiv preprint quant-ph/0406003*, 2004.
- [7] Rui Abreu, João Paulo Fernandes, Luis Llana, and Guilherme Tavares. Metamorphic testing of oracle quantum programs. In *2022 IEEE/ACM 3rd International Workshop on Quantum Software Engineering (Q-SE)*, pages 16–23. IEEE, 2022.
- [8] Martín Eugenio Avendaño González, “Apuntes computación cuántica, Álgebra Computacional, Grado Matemáticas UCM”, 22/23.
- [9] Huai Liu, Fei-Ching Kuo, Dave Towey, and Tsong Yueh Chen. 2014. How effectively does metamorphic testing alleviate the oracle problem? *IEEE Transactions on Software Engineering* 40, 1, 4–22.
- [10] Pak-Lok Poon, Fei-Ching Kuo, Huai Liu, and Tsong Yueh Chen. 2014. How can non-technical end users effectively test their spreadsheets? *Information Technology and People* 27, 4, 440–462.
- [11] Vu Le, Mehrdad Afshari, and Zhendong Su. 2014. Compiler validation via equivalence modulo inputs. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI’14)*. ACM, New York, NY, 216–226.
- [12] Christopher Lidbury, Andrei Lascu, Nathan Chong, and Alastair F. Donaldson. 2015. Many-core compiler fuzzing. In *Proceedings of the 36th ACM SIGPLAN Conference*

on Programming Language Design and Implementation (PLDI'15). ACM, New York, NY, 65–76.

- [13] John Regehr. 2014. Finding compiler bugs by removing dead code.
- [14] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26:1411–1473, 1997.
- [15] Nuno Costa, João Paulo Fernandes, and Rui Abreu. Asserting the correctness of Shor implementations using metamorphic testing. In *Proceedings of the 1st International Workshop on Quantum Programming for Software Engineering*, pages 32–36, 2022.
- [16] Matteo Paltenghi and Michael Pradel. Morphq: Metamorphic testing of the qiskit quantum computing platform, 2023.