

# 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, cada vez estamos más cerca de poder hacer un uso real de sus algoritmos. Pero esto nos genera un nuevo problema, o más bien, unas nuevas inquietudes sobre cómo vamos a programarlos, ya que cada vez son más complejos, y además cómo vamos a probar su corrección. Aquí, es donde presentamos nuestro estudio de las propiedades metamórficas de estos algoritmos, que nos van a ayudar a introducir el *testing* metamórfico, siendo esta una de las posibilidades que tenemos para alcanzar nuestro objetivo.

**Palabras clave:** Computación cuántica, qiskit, propiedades metamórficas

# Abstract

The quick development that quantum computing has been experiencing in recent years is opening up the usage of its algorithms. However, this is generating new questions, such as how we are going to program or prove the correctness of these algorithms, which are becoming increasingly complex over time. We will introduce metamorphic testing as one of the testing methods to support us in achieving this goal.

**Keywords:** Quantum computing, qiskit, metamorphic rules.

# Índice general

<b>Índice</b>	<b>1</b>
<b>1. Introducción</b>	<b>2</b>
1.1. Metodología . . . . .	3
1.2. Objetivos . . . . .	4
<b>2. Antecedentes</b>	<b>5</b>
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
<b>3. Algoritmos cuánticos</b>	<b>23</b>
3.1. Suma . . . . .	23
3.2. Deutsch . . . . .	25
3.3. Deutsch-Jozsa . . . . .	30
3.4. Bernstein-Vazirani . . . . .	34
3.5. Simon . . . . .	37
<b>4. Propiedades metamórficas</b>	<b>40</b>
4.1. Deutsch-Jozsa . . . . .	40
4.2. Bernstein-Vazirani . . . . .	42
4.3. Simon . . . . .	46
<b>5. Conclusiones y trabajo futuro</b>	<b>50</b>
5.1. Conclusiones . . . . .	50
5.2. Trabajo futuro . . . . .	51
<b>Bibliografía</b>	<b>53</b>

# Capítulo 1

## Introducción

La memoria y el trabajo presentado a continuación, va a ser el recorrido que nos va a mostrar, desde los principio básicos de la mecánica cuántica y como directamente definen la computación cuántica, hasta una de las posibilidades que tenemos para hacer testing sobre estos programas. Antes de proceder con la forma en la que se ha trabajado y como se ha ido evolucionando y aprendiendo, veamos donde comenzó todo.

La mecánica cuántica se empezó a desarrollar en los años 20, pero no sería hasta los años 80 cuando se empezó a plantear la posibilidad de la aplicación que podía tener esta teoría sobre la computación[1].<sup>1</sup> Paul Benioff presentó en 1980 la *máquina de Turing cuántica* que utilizaba la teoría cuántica para describir un ordenador simplificado. Ya en 1984, se utilizó dicha teoría sobre protocolos criptográficos de la mano de Charlos Bennett y Gilles Brassard.

A partir de entonces ya empezaron a surgir nuevos algoritmos, principalmente para resolver el problema de oráculo. Entre ellos, los algoritmos de Deutsch, Deutsch-Jozsa, Bernstein-Vazirani y Simon. Aquí ya se puede observar la mejora en eficiencia respecto al ordenador clásico, como conjeturó Richard Feynman en 1982[2]. Se podría decir, que lo que hizo despertar al resto de la comunidad sobre la importancia que podía tener la computación cuántica, llegó de la mano de Peter Shor en 1994 con sus algoritmos que podrían permitir romper las claves de encriptación RSA y Diffie-Hellman, que aún se siguen utilizando. Desde entonces la inversión en el estudio de este campo ha ido creciendo, siendo el primero ordenador de dos qubits creado en 1998 y haciendo factible esta posibilidad. En la actualidad, IBM tiene el ordenador con mayor número de qubits, 433, el `ibm_seattle` presentado a finales de 2022 y trabajando para presentar el siguiente sistema cuántico con 1121 qubits a finales de este mismo año<sup>2</sup>. Hay que destacar que el anterior sistema solo tenía 127 qubits, ¡pero este se presentó solo hace 2 años, en 2021!

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Quantum\\_mechanics#History](https://en.wikipedia.org/wiki/Quantum_mechanics#History)

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

Con toda esta evolución que está ocurriendo a nivel del número de qubits y las mejoras que se van realizando para alcanzar mayor fiabilidad, empieza a abrir la puerta al uso real de los algoritmos y programas cuánticos. Aunque estos no llegarán 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 vamos a plantear en este trabajo es como la unión entre la computación cuántica y el *testing* metamórfico puede ayudarnos a contestar esta pregunta y a intentar buscar esa corrección o falta de errores. Donde el *testing* metamórfico es uno de los métodos usados para la búsqueda de errores desde que se presentó en 1998[3], basado en el estudio de las propiedades lógicas que se pueden obtener de los algoritmos.

## 1.1. Metodología

La metodología seguida para este trabajo digamos que se puede distinguir en 3 fases que van a ser apreciables en esta memoria. De hecho, van a corresponder una a una con los capítulos siguientes. 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]. Vamos a introducir brevemente dichas fases:

- **Antecedentes:** Esta primera fase de estudio se basa en poder avanzar desde los conocimientos que tenemos del grado, a la base útil para poder entender los algoritmos cuánticos, la obtención de las propiedades metamórficas y su utilización en el testing. Además de los textos mencionados anteriormente, de los que he ido obteniendo los resultados más teóricos, en este capítulo se han fijado los conceptos de la parte de testing y propiedades metamórficas con el artículo *Metamorphic Testing: A Review of Challenges and Opportunities*[5].
- **Programación cuántica y algoritmos:** Una vez realizada y adquirida la base para empezar a entender los algoritmos y la programación, se realizaron los primeros pasos de programación, empezando por un algoritmo tan sencillo como sería la suma y como sería mi implementación de la misma utilizando *Qiskit*. Posteriormente, fui avanzando algoritmo a 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 todos los algoritmos programados y pruebas realizadas, para

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

- **Propiedades y testing metamórfico:** Ahora que ya hemos conseguido entender e incluso programar nuestros algoritmos cuánticos, vamos a por el último eslabón de la cadena. Este es el estudio de las propiedades metamórficas sobre los algoritmos de Deutsch-Jozsa, Bernstein-Vazirani y Simon. Como apoyo para la comprensión y estudio de esta fase, nos hemos apoyado en el artículo *Metamorphic Testing of Oracle Quantum Programs*[7]. Además, esta parte del trabajo se ha realizado de forma conjunta con Rodrigo de la Nuez Moraleda, y todo lo que hemos programado y preparado para el testing de estos algoritmos se puede 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 seguida y los materiales utilizados, siguiendo el mismo esquema vamos a analizar los objetivos. Hay que 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, pero para llegar hasta aquí vamos a ir alcanzando ciertos objetivos y competencias secundarias que pretendemos adquirir en cada fase.

- **Antecedentes:** Queremos ser capaces de entender las definiciones más básicas, partiendo de los conocimientos matemáticos, principalmente del álgebra lineal. Algunos ya conocidos como que representa una matriz unitaria o hermitiana y la relación que existe entre ellas y sus operadores, o introducir nuevos conceptos, como el espacio de Hilbert y el producto tensorial, que nos servirá para generar sistemas más complejos. Además estudiaremos la parte física de computación cuántica, con sus postulados y su relación con la programación cuántica e introduciremos los elementos básicos de dicha programación y 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. Así como entender como 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 poder entender y trabajar con propiedades metamórficas y en particular, su aplicación a la computación cuántica. Para ello, vamos a hacer un breve repaso a conceptos básicos de álgebra lineal en matemáticas, los postulados de la mecánica cuántica y una iniciación a la computación cuántica y el testing metamórfico.

Esta sección, que podría ser una simple continuación de la introducción, va a ser más extensa de lo que se podría esperar. Ya que para trabajar de forma cómoda, sobre el tema a tratar, necesitamos un salto en conocimientos que se han tenido que adquirir.

### 2.1. Introducción matemática

Para poder desarrollar y entender la mecánica cuántica, que presentaré a continuación, la programación cuántica y en particular, sus algoritmos, vamos a necesitar cierta base matemática y de notación. Quizás, las definiciones que siguen este párrafo pueden parecer aleatorias, aunque todo cobrará sentido conforme vayamos profundizando en la mecánica y programación cuántica.

Definimos un **espacio de Hilbert**,  $\mathcal{H}$ , como un  $\mathbb{C}$ -espacio vectorial dotado de un producto interno que es completo. En particular, como vamos a tratar solo  $\mathbb{C}$ -espacios vectoriales con producto interno finito, este 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  $\mathcal{A}^* : \mathcal{H} \rightarrow \mathcal{H}$  talque  $\langle \mathcal{A}v, w \rangle = \langle v, \mathcal{A}^*w \rangle$ .
- 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 con esta sección vamos a ver una manera de combinar dos espacios vectoriales, en particular, dos espacios de Hilbert que nos permitirá unir dos sistemas cuánticos, esta operación es el producto tensorial.[4]<sup>1</sup>

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

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

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

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:

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

---

<sup>1</sup>[https://es.wikipedia.org/wiki/Producto\\_tensorial](https://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 productores 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 ayuda a 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 vamos a tener 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 comienzo de la computación cuántica fue el desarrollo de la física cuántica. Para poder entender mejor que variaciones e implicaciones tiene, vamos a ver los postulados de la mecánica cuántica y sus diferencias con la mecánica Newtoniana. Aquí encontrará sentido la base matemática presentada en la sección 2.1.

Para empezar, en mecánica clásica, un sistema de  $N$  partículas queda definido por 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 evolución de un sistema viene determinado por sus postulados[4][8] que veremos a continuación <sup>2</sup>, así como una de las posibles interpretaciones que cada uno podría tener. Estos postulados nos ayudaran posteriormente a fijar 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 del los espacios de cada componente del sistema. Es decir, si tuviéramos  $n$  componentes, el espacio total sería  $|\varphi_1\rangle \otimes |\varphi_2\rangle \otimes \dots \otimes |\varphi_n\rangle = |\varphi_1\varphi_2\dots\varphi_n\rangle$ , de forma notacional.

### 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.  $|\varphi(t)\rangle = U(t; t_0)|\varphi(t_0)\rangle$ .

---

<sup>2</sup>[https://en.wikipedia.org/wiki/Mathematical\\_formulation\\_of\\_quantum\\_mechanics#Postulates\\_of\\_quantum\\_mechanics](https://en.wikipedia.org/wiki/Mathematical_formulation_of_quantum_mechanics#Postulates_of_quantum_mechanics)

- **Cuarto postulado:** Evolución probabilística, tenemos observador.
- **Primer apartado:** Cada medida  $\mathcal{A}$  esta descrita por un operador hermitiano  $A$  que actúa sobre  $\mathcal{H}$ , decimos que este operador es 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  $a_n$ , entonces el estado del sistema colapsa a la proyección normal de  $|\varphi(t)\rangle$  en el subespacio de autovectores asociado a  $a_n$ ,  $P_{|v_n\rangle}|\varphi(t)\rangle$ .

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

## 2.3. Programación cuántica, Qiskit

La programación cuántica se basa en la creación de un circuito o algoritmo cuántico, normalmente representado geométricamente, donde se realizan operaciones (operadores unitarios) sobre los distintos qubits, así como sus mediciones[1].

Para realizar nuestros programas cuánticos y simulaciones nos vamos a apoyar en *Qiskit*, es un paquete de desarrollo libre creado por IBM para crear y manipular programas cuánticos así como realizar simulaciones<sup>3</sup>. Ya sean teóricas o conectando nuestros programas con los ordenadores cuánticos de IBM. Esto nos dará unos resultados más realistas donde podremos apreciar el ruido que hay actualmente en estos ordenadores.

Se programará en Qiskit, una librería de Python, en el que se permite definir y utilizar los circuitos. Además, para una mejor visualización de lo que representa el código utilizaré jupyter notebook. Otra opción sería generar los circuitos de forma geométrica en IBM<sup>1</sup>.

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

---

<sup>3</sup><https://qiskit.org/>

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

<sup>5</sup><https://quantumai.google/cirq>

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

Como mencionamos anteriormente los postulados cuánticos, ordenados de acorde a la utilización en este texto, nos van a permitir sentar 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 cuadrado de los coeficientes determinan la probabilidad de aparición y por lo tanto la suma de todas la probabilidad de todas las posibilidades 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 <sup>7</sup>. 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.

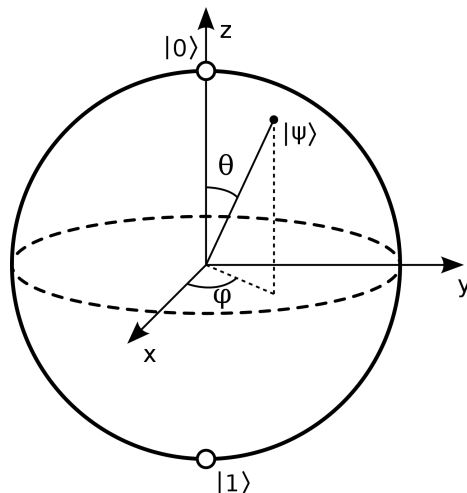


Figura 2.1: Esfera de Bloch

<sup>7</sup>[https://en.wikipedia.org/wiki/Bloch\\_sphere#/media/File:Bloch\\_sphere.svg](https://en.wikipedia.org/wiki/Bloch_sphere#/media/File:Bloch_sphere.svg)

Veamos que es lo que ocurre ahora cuando en vez de un único qubit, tenemos varios y la relación que hay entre ellos. Para simplificarlo, tomemos el más simple, el sistema con 2 qubits. Si recordamos el postulado 2 nos determinaba como interaccionaban estos dos qubits, como el sistema compuesto por ambos era su producto tensorial. 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,  $a_{00} + a_{01} + a_{10} + a_{11} = 1$ . Este resultado se puede obtener del producto de Kronecker, 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 como operamos en el sistema, vamos a introducir un último concepto, el enredo cuántico o **entanglement**. Esta es una propiedad de la mecánica cuántica que aparece en un sistema, se puede decir que 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 dicho 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 va a permitir obtener una idea.

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

$$(Bell\ state)\sqrt{2} = 1|00\rangle + 0|01\rangle + 0|10\rangle + 1|11\rangle$$

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

$$\begin{aligned} (a_0|0\rangle + a_1|1\rangle) \otimes (a'_0|0\rangle + a'_1|1\rangle) &= a_0a'_0|00\rangle + a_0a'_1|01\rangle + a_1a'_0|10\rangle + a_1a'_1|11\rangle \Rightarrow \\ &\Rightarrow a_0a'_0 = a_1a'_1 = 1 \text{ y } a_0a'_1 = a_1a'_0 = 0 \end{aligned}$$

Pero este sistema de ecuaciones no tiene solución, por lo que  $(Bell\ state)\sqrt{2}$  no se puede expresar como producto tensorial y en particular, *Bell state* 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 se mida el primer qubit ya tendremos el valor del segundo sin haberlo medido <sup>8</sup>. Podemos encontrar otros ejemplos y preguntas muy interesantes sobre *entanglement* en la página de IBM, donde estudia distintas posibilidades con la puerta CNOT, que presentaremos a continuación, y estados interesantes como *Bell state*, *GHZ states* and *W states*.

Otro de los resultados interesantes es el Teorema de no Clonación, por el cual nos indica que no podemos copiar el estado de un qubit desconocido a otro qubit. Esto se puede entender, ya que si observáramos el qubit estaríamos realizando una medición y el estado colapsaría.

**Teorema 2.1 (No clonación)** [4]: Sea  $\mathcal{H}$  un espacio de Hilbert en  $\mathbb{C}$  y sean  $|\varphi\rangle, |\psi\rangle \in \mathcal{H}$ . Entonces,  $\nexists U$  unitario sobre  $\mathcal{H} \otimes \mathcal{H}$  talque  $U|\varphi\rangle|\psi\rangle = |\varphi\rangle|\varphi\rangle$ .<sup>9</sup>

### 2.3.1. Puertas y circuitos cuánticos

Partimos ahora del tercer postulado, en particular el apartado 2. Se podría ver que existe una correspondencia 1 a 1 entre el Hamiltoniano  $H$ , por ser hermitiano, y un operador unitario  $U$ . Estos operadores unitarios serán nuestras **puertas cuánticas** que vamos a usar 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, por lo cual no rompen la condición de normalización de la definición de qubit.

En programación cuántica estos operadores unitarios pueden crearse directamente con una matriz, que cumpla las condiciones necesarias, aunque habitualmente utilizaremos puertas cuánticas, operadores, específicas que son las utilizadas en los ordenadores cuánticos reales[8]. Veamos cuales son las puertas cuánticas mas útiles para un qubit:[4] <sup>10</sup>

- **Puertas de Pauli:** Estas puertas son la más básicas y nos van a permitir, a excepción de la identidad, realizar rotaciones de  $\pi$  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  $2 \times 2$ .

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

<sup>9</sup>[https://es.wikipedia.org/wiki/Teorema\\_de\\_no\\_clonaci%C3%B3n](https://es.wikipedia.org/wiki/Teorema_de_no_clonaci%C3%B3n)

<sup>10</sup>[https://en.wikipedia.org/wiki/Quantum\\_logic\\_gate](https://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 tiene su propia notación:

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

A su vez, al igual que las matrices de Pauli, la matriz de Hadamard realiza una rotación de  $\pi$  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  $\theta$  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 referimos a la fase general 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  $\alpha, \beta, \gamma$  y  $\delta$  tal que:

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

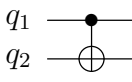
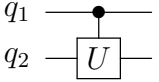
Existe un resultado análogo para X-Y. Este teorema va a permitir descomponer cualquier operador unitario en estas rotaciones y así, los ordenadores cuánticos actuales, pueden procesar cualquier circuito independientemente de las diferencias entre puertas que componen el circuito cuántico y las puertas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


Veamos 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  $U$ .

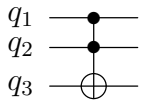
La representación de CNOT en un circuito es: . En general, .

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

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

Hay que aclarar que esta puerta no copia los estados de los qubits, si no 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 . Queda determinada por,

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

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**<sup>11</sup> 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, si lo es. Para solucionar este problema, nos apoyamos en el teorema de *Solovay-Kitaev*<sup>12</sup>, 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 presentado como por ejemplo, los subíndices indicarán a que qubit se aplican en caso de ser necesario:


<sup>11</sup>[https://en.wikipedia.org/wiki/Quantum\\_logic\\_gate#Universal\\_quantum\\_gates](https://en.wikipedia.org/wiki/Quantum_logic_gate#Universal_quantum_gates)

<sup>12</sup>[https://en.wikipedia.org/wiki/Solovay%E2%80%93Kitaev\\_theorem](https://en.wikipedia.org/wiki/Solovay%E2%80%93Kitaev_theorem)

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

Alguna de estas relaciones serán observadas al implementar los algoritmos.

Ahora bien, todavía no hemos observado el sistema, solo hemos ido realizando operaciones unitarias, es decir, seguimos en un sistema determinista pero sin conocer realmente lo que está ocurriendo. Si recordamos otro postulado de la física cuántica, en particular el cuarto, se refería a la evolución dinámica del sistema cuando era observado. Aquí dejaremos la evolución determinista y pasaremos a la probabilística. Para poder entender que ha ocurrido en nuestro sistema y obtener resultados, vamos a necesitar medir.

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, obtendrá uno de los elementos donde proyectamos con cierta probabilidad. Que 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  $|1\rangle$  será  $|b|^2$ . 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 almacenar esta información utilizaremos bits clásicos, por eso podremos observar como la medición se representa con la puerta anterior y la caída hacia el bit clásico donde se almacene.

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.

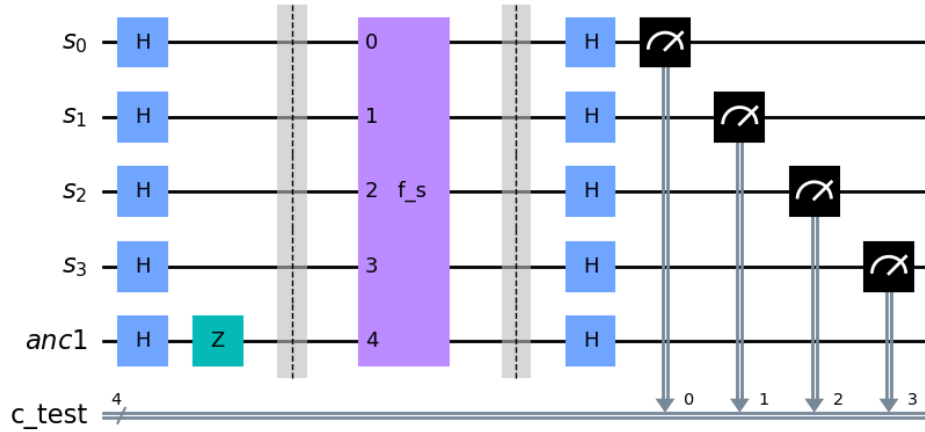


Figura 2.2: Circuito, algoritmo BV para  $s$  de longitud 4

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 esta puerta tenemos puertas más pequeñas como las presentadas anteriormente en este mismo apartado. 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<sup>13</sup>:

- **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, los errores de cada qubit así como la estructura del ordenador, 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.

---

<sup>13</sup>[https://qiskit.org/documentation/qc\\_intro.html](https://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 lo que 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 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.

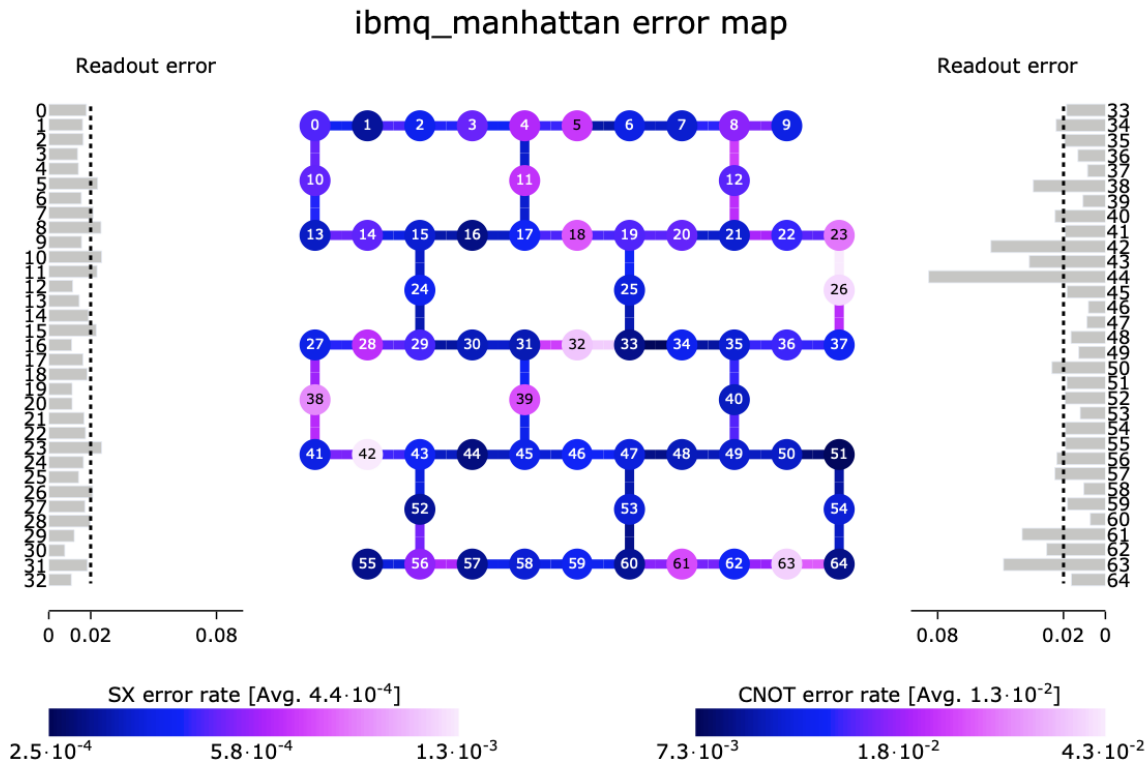


Figura 2.3: Mapa de errores del `ibmq_manhattan`

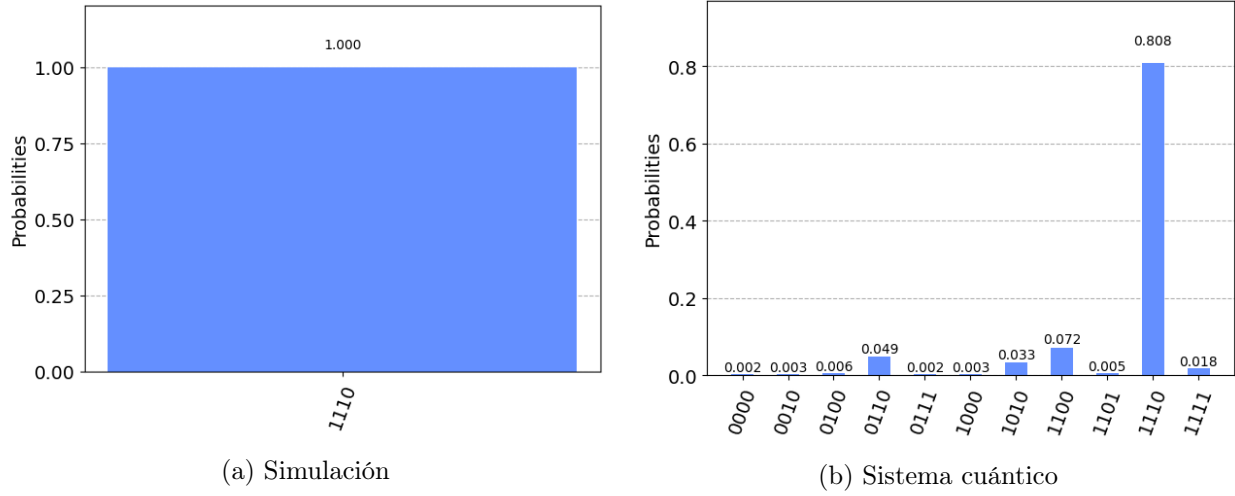


Figura 2.4: Diferencias en resultados de ejecución del algoritmo de Bernstein-Vazirani

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*<sup>14</sup>. El código que utilizaría *Qiskit* para la ejecución del circuito presentado anteriormente lo podemos encontrar en el trabajo generado en el sistema cuántico de IBM<sup>15</sup>. Además, vamos a poder observar en la Figura 2.5 el circuito transformado acorde a las puertas de las que dispone el sistema cuántico preparado para la ejecución. Este circuito es el análogo a la Figura 2.2 con resultados en la Figura 2.4b.

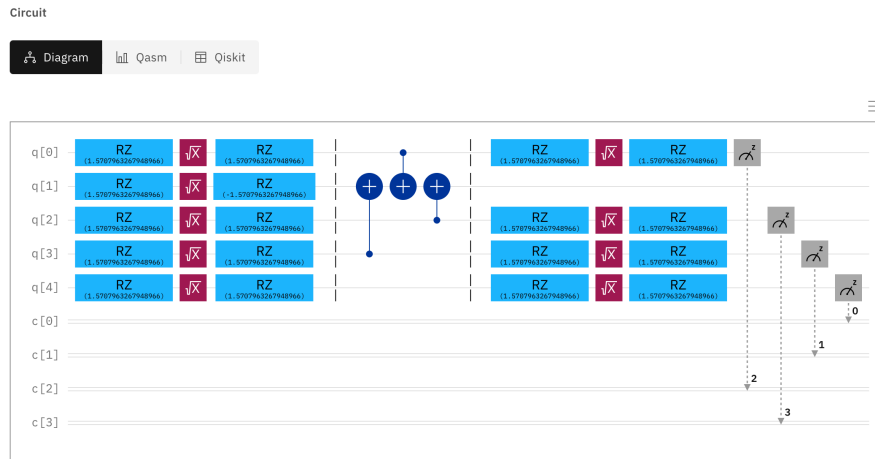


Figura 2.5: Circuito del sistema ibmq\_lima para BV, Figura 2.2

<sup>14</sup><https://qiskit.org/documentation/stubs/qiskit.compiler.assemble.html>

<sup>15</sup><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 empezamos 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.** Aun se sigue estudiando que 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, aun no hay un acuerdo sobre una estructura definida y formal que nos proporcione seguridad en nuestra pruebas y englobe a todas las posibilidades que tenemos dentro de las MR. Aunque, si bien es cierto, ya ha ayudado a 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 encontré 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. Trataremos en más profundidad estas posibilidades en la sección 5.2 de posibles trabajos futuros.

# 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 o avanzar.

Todo el desarrollo de estos algoritmos se pueden encontrar en los libros principales [4][1]. Presentaremos a continuación todos los algoritmos finales que nos permiten obtener nuestros objetivos, si bien es cierto, solo vamos a presentar el camino completo de creación del algoritmo de Deutsch por su simplicidad. Esto se debe a que para alcanzar nuestro objetivo debemos ir haciendo modificaciones y cálculos sobre nuestros algoritmos hasta dar con la combinación correcta de puertas que nos permita resolverlo.

Esta sección va a seguir prácticamente la misma estructura para cada apartado, a excepción de la suma. Empezará con una introducción, seguida de la exposición del problema a resolver. Entonces nos dispondremos a presentar el algoritmo que lo resuelve, o como lo creamos, y las pruebas realizadas. Es necesario recordar que toda la programación realizada sobre 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 bit ancilla como longitud tenga la cadena, para que almacene el resultado, así nos aseguramos que no se modifiquen los valores originales de entrada y 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.

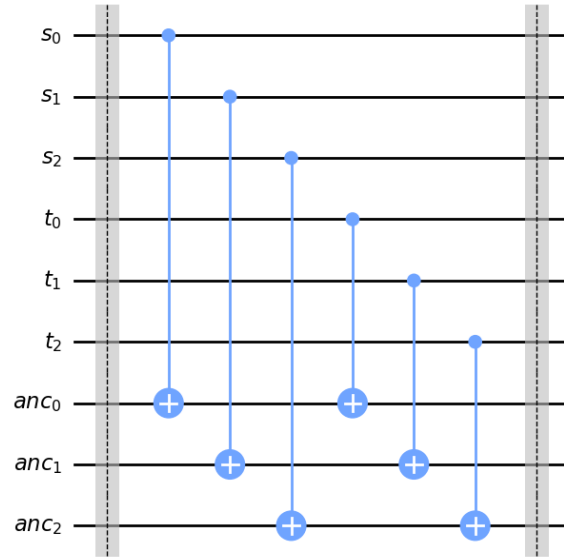


Figura 3.1: Suma binaria bit a bit, para cadenas  $s$  y  $t$  de longitud 3

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 esto vamos a utilizar la puerta *CCNOT*, que nos va a permitir modificar el siguiente ancilla qubit si fuera necesario. De esta manera nos aseguramos de realizar la suma correctamente junto al uso de la puerta *CNOT*.

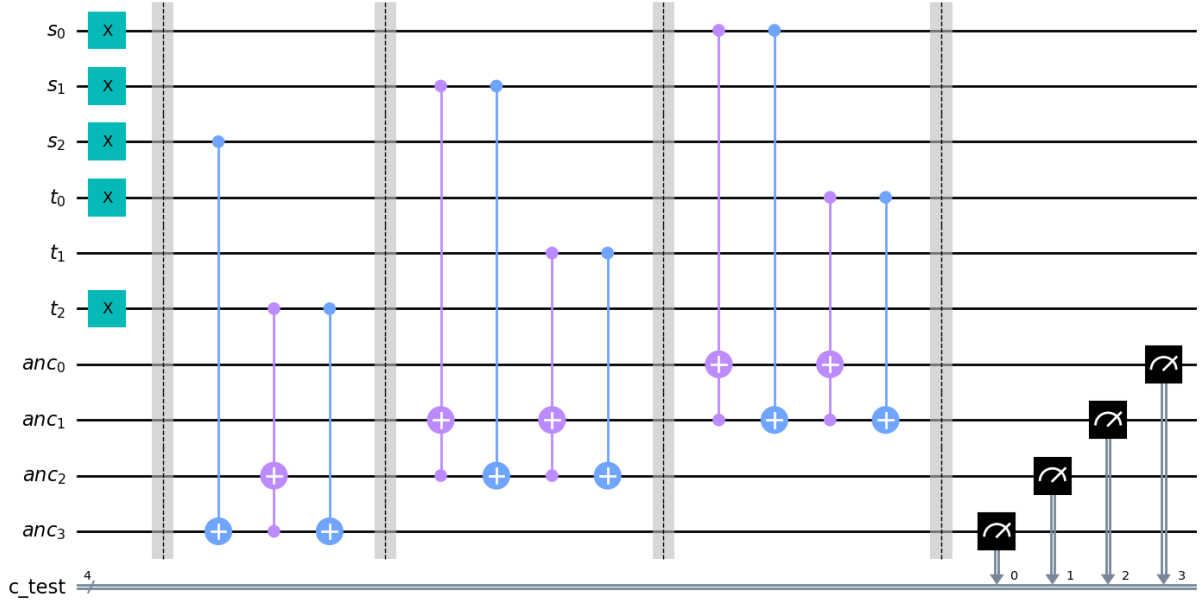


Figura 3.2: Suma de las cadenas 111 y 101 de longitud 3

Aquí se puede apreciar la suma completa, las puertas  $X$  antes de las primera barrera codifican las cadenas binarias y luego se sigue el procedimiento explicado, excepto por 2 casos. El primero es que la primera puerta  $CCNOT$  no está, esto se debe a que no es necesaria ya que  $q_9$  se inicializa en 0, y por lo tanto la puerta no realizaría ningún cambio sobre  $q_8$ . El segundo es la inversión en la medición de los qubits, esta decisión se debe a una característica que tiene Qiskit con los qubits, en especial con la base que utiliza. De esta manera nos aseguramos el orden de los bits correctos, como se puede ver en el documento del repositorio. Esta peculiaridad se estudiará más a fondo con el algoritmo de Deutsch-Jozsa.

**Simulaciones:** La simulación y código se encuentran en el repositorio<sup>1</sup>.

## 3.2. Deutsch

El algoritmo de Deutsch[1] fue propuesto por David Deutsch en 1985, siendo este uno de los primeros algoritmos propuestos y en sí, el que se podría entender como uno de los problema más simples. Sea  $f : \{0, 1\} \rightarrow \{0, 1\}$ , diremos que  $f$  es **balanceada** si  $f(0) \neq f(1)$  y diremos que es **constante** si  $f(0) = f(1)$ .

<sup>1</sup><https://github.com/sinugarc/TFG/blob/main/SumaNat.ipynb>

**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:

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

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

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

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:

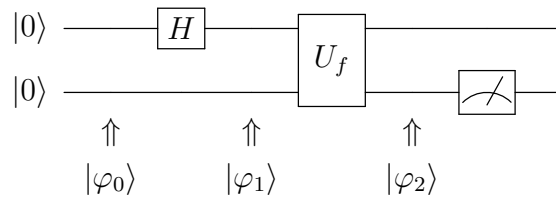
$$\begin{array}{c}
|00\rangle \quad |01\rangle \quad |10\rangle \quad |11\rangle \\
\begin{array}{c} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{array} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{array}$$

Se podría comprobar que esta matriz es su propia adjunta, por lo que es invertible y unitaria. Por lo que 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, esta va a ser el único algoritmo al que le vamos a seguir 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  $\varphi_i$  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  $i$ :

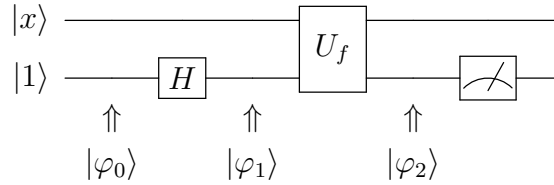
- $|\varphi_0\rangle = |0\rangle \otimes |0\rangle = |0, 0\rangle$
- $|\varphi_1\rangle = (H \otimes I)|0, 0\rangle = \left[ \frac{|0\rangle + |1\rangle}{\sqrt{2}} \right] |0\rangle = \frac{|0, 0\rangle + |1, 0\rangle}{\sqrt{2}}$
- $|\varphi_2\rangle = \frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}}$

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  $|\varphi_2\rangle$  :

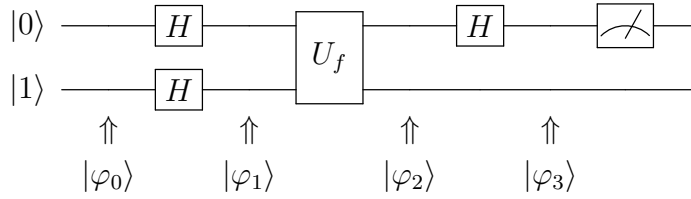
$$|\varphi_2\rangle = |x\rangle \left[ \frac{|0 \otimes f(x)\rangle - |1 \otimes f(x)\rangle}{\sqrt{2}} \right] = \begin{cases} |x\rangle \left[ \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right] & \text{si } f(x) = 0 \\ |x\rangle \left[ \frac{|0\rangle + |1\rangle}{\sqrt{2}} \right] & \text{si } f(x) =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

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 vamos a poner ambos en estado de superposición, con el *input*  $|0, 1\rangle$ . Además, vamos a medir 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 (link). 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 lo creo como operador a partir de matriz.

**Simulaciones:** Las simulaciones y código se encuentran en el repositorio<sup>2</sup>.

### 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.

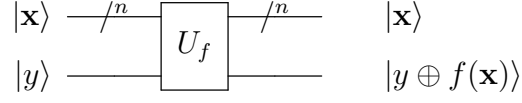
**Problema:** Dada una función  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  balanceada o constante, la cual no podemos observar su definición. Queremos determinar si esta función es constante o balanceada.

**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.

---

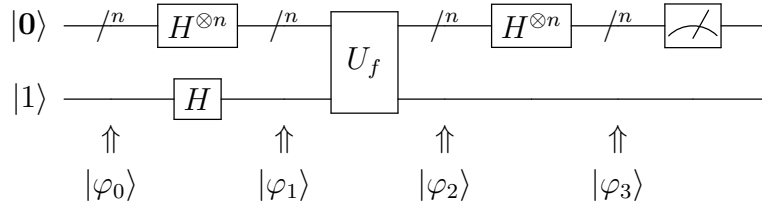
<sup>2</sup><https://github.com/sinugarc/TFG/blob/main/Deutsch.ipynb>

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:  $(H^{\otimes n} \otimes I) U_f (H^{\otimes n} \otimes H) |\mathbf{0}, 1\rangle$

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 en  $\mathbb{Z}_2$ :

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

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 si 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 tenemos que superponer la superposición, pero en este caso no es la inversa ya que se aplica sobre otros estados  $|\mathbf{z}\rangle$ :

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

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  $|\mathbf{0}\rangle$  al realizar la medición. Esto es análogo a si bien en el algoritmo anterior nos interesaba medir cuando  $|\mathbf{z}\rangle = |\mathbf{0}\rangle$ , 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 de nuestras posibilidades:

- $f$  es constante:  $\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} |\mathbf{0}\rangle}{2^n} = \frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} |\mathbf{0}\rangle}{2^n} = \frac{(\pm 1) 2^n |\mathbf{0}\rangle}{2^n} = (\pm 1) |\mathbf{0}\rangle$
- $f$  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 los resultados no son concluyentes. Esto se debe a que cualquier valor que nos devuelva el algoritmo tiene un interpretación  $|\mathbf{0}\rangle$   $f$  constante y si no,  $f$  es balanceada. Es decir, si tenemos una función  $f$  que 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<sup>3</sup>, 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<sup>4</sup> 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 esta programado, así como pruebas del algoritmo de DJ con la misma matriz.

Si bien es cierto que todos aquellos algoritmos que tiene programado el oráculo con el procedimiento de Qiskit de forma balanceada su resultado es el ket  $|1\rangle$ , esto no tiene porque ser cierto. Si recordamos el análisis final del algoritmo, si obtenemos  $|0\rangle$  entonces  $f$  es constante y si no,  $f$  es balanceada. Por lo que si miramos el algoritmo programado con un operador matricial, podemos observar como el resultado no es ni  $|0\rangle$ , ni  $|1\rangle$ , si no que obtenemos  $|01\rangle$ .

### 3.4. Bernstein-Vazirani

El algoritmo de Bernstein-Vazirani, fué presentado por Ethan Bernstein and Umesh Vazirani[14]. Se puede entender que este algoritmo es una generalización más del algoritmo de Deutsch-Jozsa <sup>5</sup>.

**Problema:** Sea  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ , sea  $\mathbf{s} = s_0s_1\dots s_{n-1}$ , por hipótesis  $f(\mathbf{x}) = \mathbf{s} \cdot \mathbf{x} \bmod 2 = s_0x_0 \oplus s_1x_1 \oplus \dots \oplus s_{n-1}x_{n-1}$ . 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  $\mathbf{s}$ .

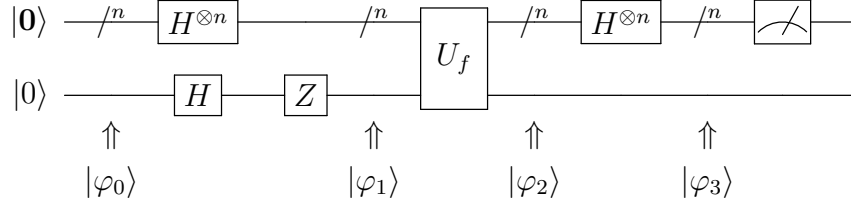
---

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

<sup>4</sup>[https://github.com/sinugarc/TFG/blob/main/Deutsch\\_Jozsa.ipynb](https://github.com/sinugarc/TFG/blob/main/Deutsch_Jozsa.ipynb)

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

### Algoritmo de Bernstein-Vazirani:



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

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  $|\mathbf{z}\rangle$ :

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

Veamos que esta ocurriendo en ese primer bloque de qubits, que será al que apliquemos la medición, si bien en el algoritmo anterior nos interesaba medir cuando  $|\mathbf{z}\rangle = |\mathbf{0}\rangle$ , pero ahora queremos obtener la cadena  $\mathbf{s}$ , por lo que vamos a ver que sucede cuando  $|\mathbf{z}\rangle = |\mathbf{s}\rangle$ . Es decir, queremos analizar la amplitud del estado.

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

¿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. Sí el valor es 0, entonces pondremos una puerta identidad en la línea del qubit de esa posición para indicar que no se hace nada 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. Estos ejemplo se pueden encontrar en el archivo de BV sin puertas, la función `bv_fgenerator(qc, n0, s)`, veamos un par de ejemplos ahora:

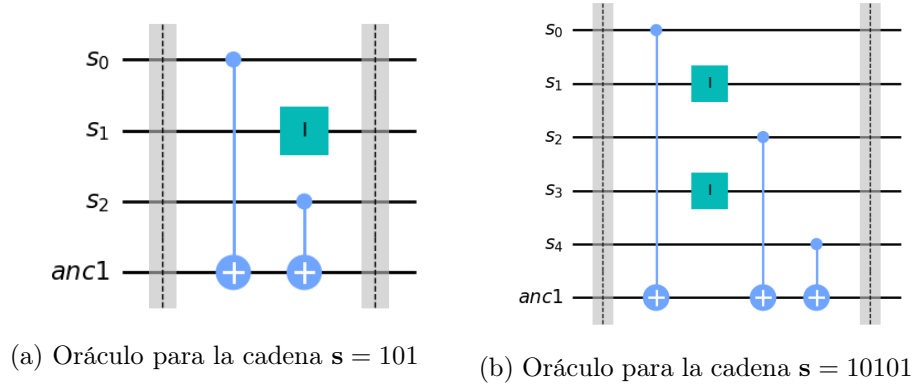


Figura 3.3: Ejemplo de oráculos para el algoritmo de BV

**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 repositorio<sup>6</sup>.

<sup>6</sup>[https://github.com/sinugarc/TFG/blob/main/Bernstein\\_Vazirani.ipynb](https://github.com/sinugarc/TFG/blob/main/Bernstein_Vazirani.ipynb)

### 3.5. Simon

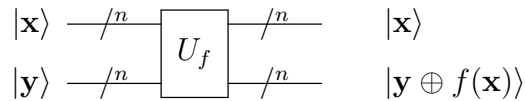
El algoritmo de Simon, o de periodicidad de Simon, va a ser el primero que no va a usar únicamente computación cuántica, si no que la combina con computación clásica al final del mismo. Veremos cual es el problema y la parte cuántica de la solución, para así ver como procedemos posteriormente hasta encontrar 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  $\mathbf{c} = c_0c_1\dots c_{n-1}$  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  $f$ .

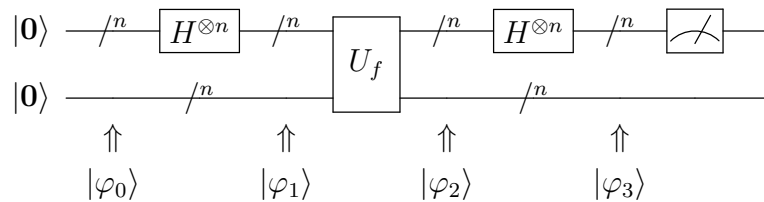
**Problema:** Dada una función  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  con periodo  $\mathbf{c} = c_0c_1\dots c_{n-1}$ . Queremos determinar cual es el periodo de  $f$  teniendo en cuenta que no conocemos su definición.

**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  $(\pm 1)$ , 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 Simon<sup>7</sup>.

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

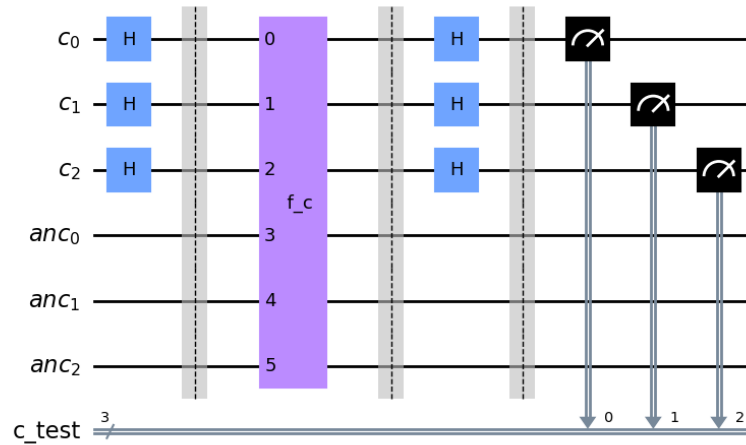


Figura 3.4: Circuito, algoritmo Simon con  $\mathbf{c} = 100$

Lo hemos simulado y ejecutado en un sistema cuántico de IBM y estos son los resultados. 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.

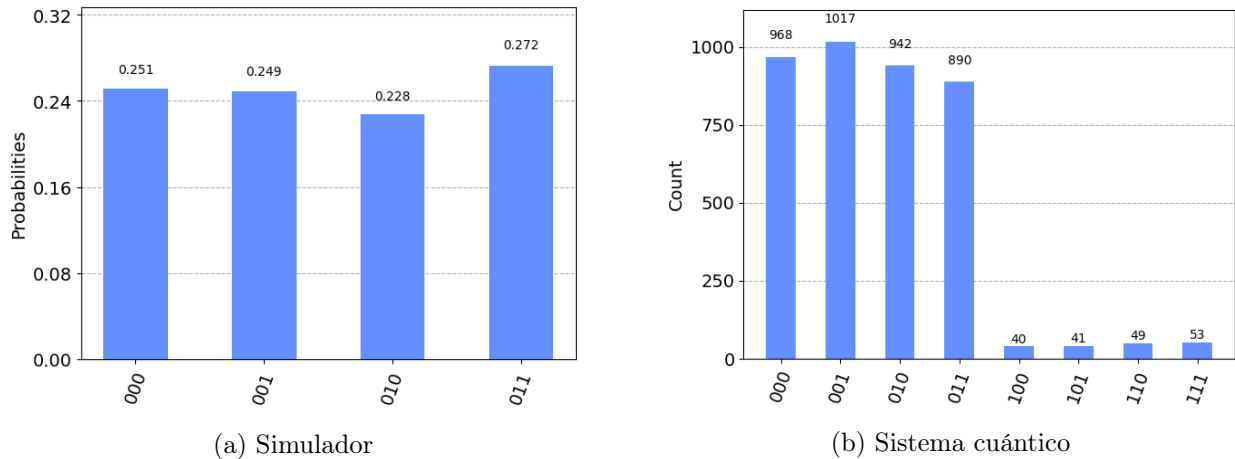


Figura 3.5: Resultados del algoritmo Simon para la cadena  $\mathbf{c} = 100$ .

<sup>7</sup>[https://github.com/rodelanu/TFG/blob/main/3\\_Simon\\_Rules.ipynb](https://github.com/rodelanu/TFG/blob/main/3_Simon_Rules.ipynb)

# Capítulo 4

## Propiedades metamórficas

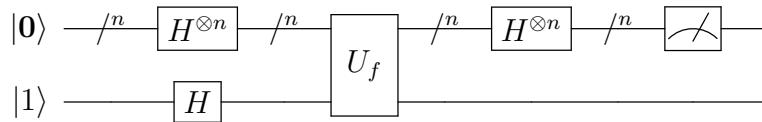
En este último capítulo antes de la conclusión queremos presentar al lector con el objetivo principal de este trabajo. Queremos finalmente unir las propiedades metamórficas con los algoritmos que hemos estudiado en el capítulo 3, si bien es cierto solo hemos conseguido estudiar los tres algoritmos que se presentan a continuación. Presentaremos cuales son las propiedades obtenidas, así como de donde las hemos obtenido, para luego pasar a la programación y los circuitos que hemos preparado para poder realizar dichas pruebas.

### 4.1. Deutsch-Jozsa

En esta sección vamos a estudiar que reglas hemos obtenido del algoritmo de Deutsch-Jozsa y como 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 que reglas hemos obtenido y como las vamos a aplicar:

**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:** Vamos a comparar los resultados entre la simulación  $P(f)$  con  $P(f \circ g)$ . 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  $\mathbf{x}$ . Este será el circuito para la  $g$  definida

**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 repositorio<sup>1</sup>, 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.

---

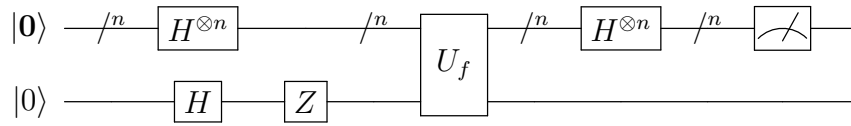
<sup>1</sup>[https://github.com/rodelanu/TFG/blob/main/1\\_Deutsch\\_Jozsa\\_Rules.ipynb](https://github.com/rodelanu/TFG/blob/main/1_Deutsch_Jozsa_Rules.ipynb)

## 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 lo visto cuando se estudió el algoritmo de BV.

**Problema:** Sea  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ , sea  $\mathbf{s} = s_0 s_1 \dots s_{n-1}$ , 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, pero eso no nos asegura que encontrar un fallo sea más fácil, por el contrario, puede ser incluso más complejo programar un regla general. Es cierto que este caso es muy simple, aunque la comprobación para este caso 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.

Ahora volvamos a 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  $s \oplus s1 = |1\rangle$ .

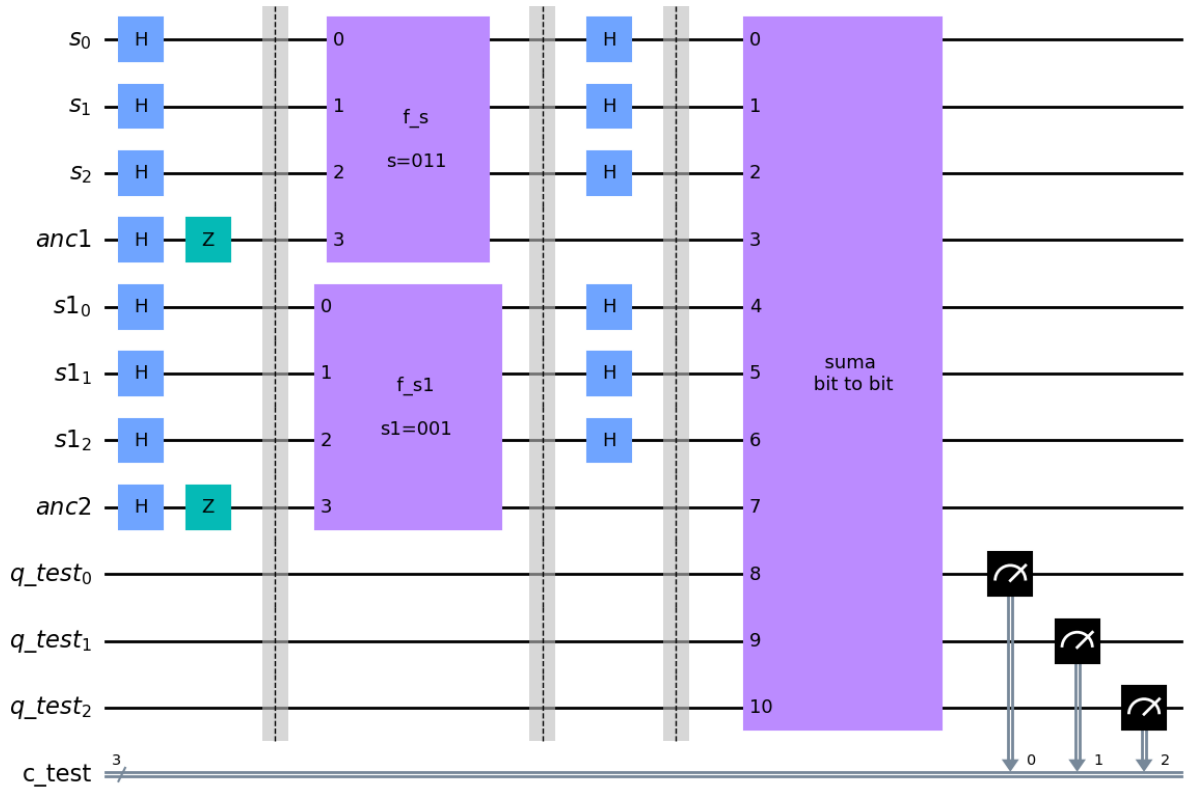


Figura 4.1: Circuito para la regla II de BV

**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 input*.

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*.

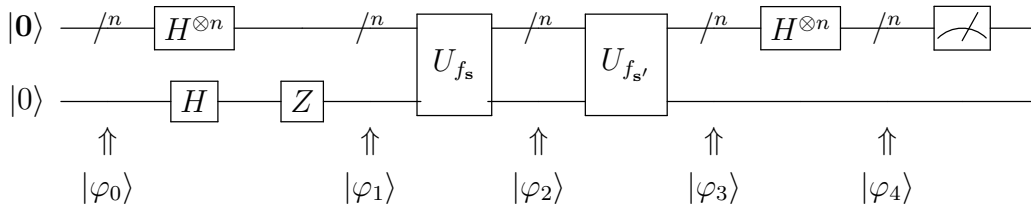
El circuito utilizado se muestra en la Figura 4.1. Es el circuito creado para las cadenas  $\mathbf{s} = 011$  y  $\mathbf{s}' = 001$ , las cuales se generan de manera aleatoria una vez determinada una longitud  $n$  que se pasa como parámetro a la función.

**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:

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

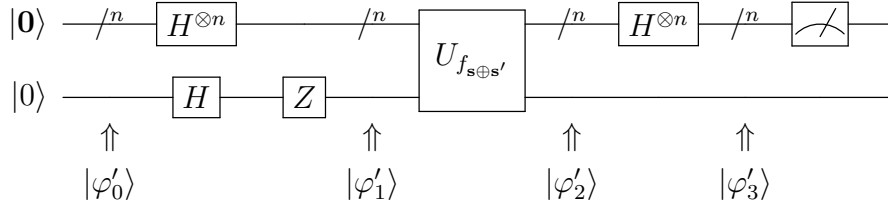
Veamos como quedaría el circuito con la concatenación, que incluye este estado:



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 reducido la complejidad de las comparaciones, para 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,



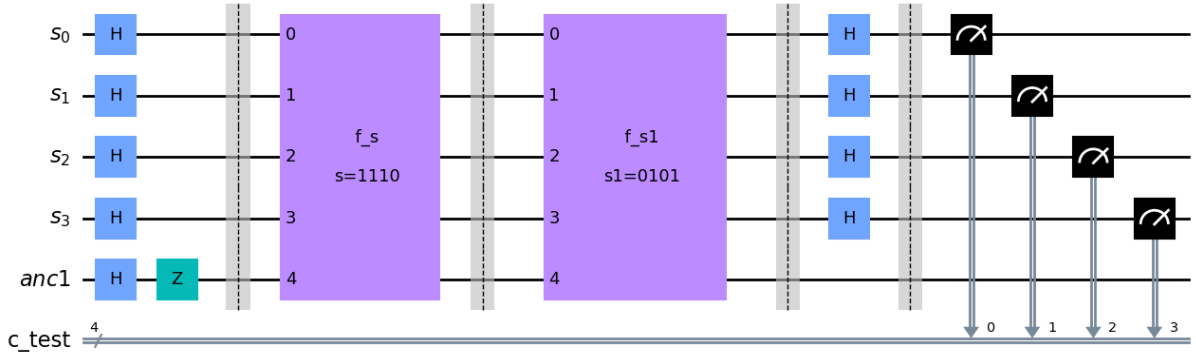


Figura 4.2: Circuito para la regla III de BV

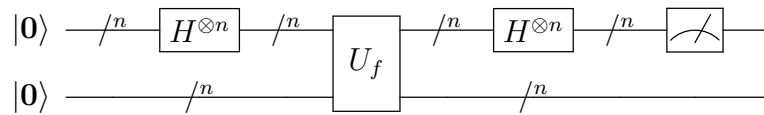
Los circuitos y pruebas realizadas se puede encontrar en el archivo de BV<sup>2</sup> del repositorio común.

### 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  $\langle \mathbf{z}, \mathbf{c} \rangle = 0$ .

<sup>2</sup>[https://github.com/rodelanu/TFG/blob/main/2\\_Bernstein\\_Vazirani\\_Rules.ipynb](https://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  $S$  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:

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

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

- **Elemento neutro:** Veamos que  $\mathbf{0}$  es elemento neutro, ya se probó en  $S_{\mathbf{c}} \neq \emptyset$  que  $\mathbf{0} \in S_{\mathbf{c}}$ ,  $\forall \mathbf{c} \in \{0, 1\}^n$ . Dada  $\mathbf{z} \in S_{\mathbf{c}}$ ,  $\mathbf{z} \oplus \mathbf{0} = \mathbf{z} = \mathbf{0} \oplus \mathbf{z}$ , se verifica  $\forall \mathbf{z} \in \{0, 1\}^n$ .
- **Elemento inverso:** Queremos ver que si  $\mathbf{z} \in S$  existe  $\mathbf{z}^{-1} \in S$  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  $\mathbf{z} \in S \Rightarrow \mathbf{z}^{-1} = \mathbf{z} \in S$  tal que  $\mathbf{z} \oplus \mathbf{z}^{-1} = \mathbf{z} \oplus \mathbf{z} = \mathbf{0}$ .
- **Asociatividad:** Sean  $\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3 \in S_{\mathbf{c}}$ ,  $(\mathbf{z}_1 \oplus \mathbf{z}_2) \oplus \mathbf{z}_3 = \mathbf{z}_1 \oplus (\mathbf{z}_2 \oplus \mathbf{z}_3)$  al ser  $\oplus$  asociativa entonces se cumple la igualdad anterior.

Por lo que ya hemos probado que  $\forall \mathbf{c} \in \{0, 1\}^n$ ,  $(S_{\mathbf{c}}, \oplus)$  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 que nosotros hemos tomado  $\mathbf{c} = |001\rangle$  para dibujar los circuitos y  $\mathbf{c} = |0001\rangle$  para la obtención de soluciones, 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 manera más sencilla de manera 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 Simon<sup>3</sup>.

---

<sup>3</sup>[https://github.com/rodelanu/TFG/blob/main/3\\_Simon\\_Rules.ipynb](https://github.com/rodelanu/TFG/blob/main/3_Simon_Rules.ipynb)

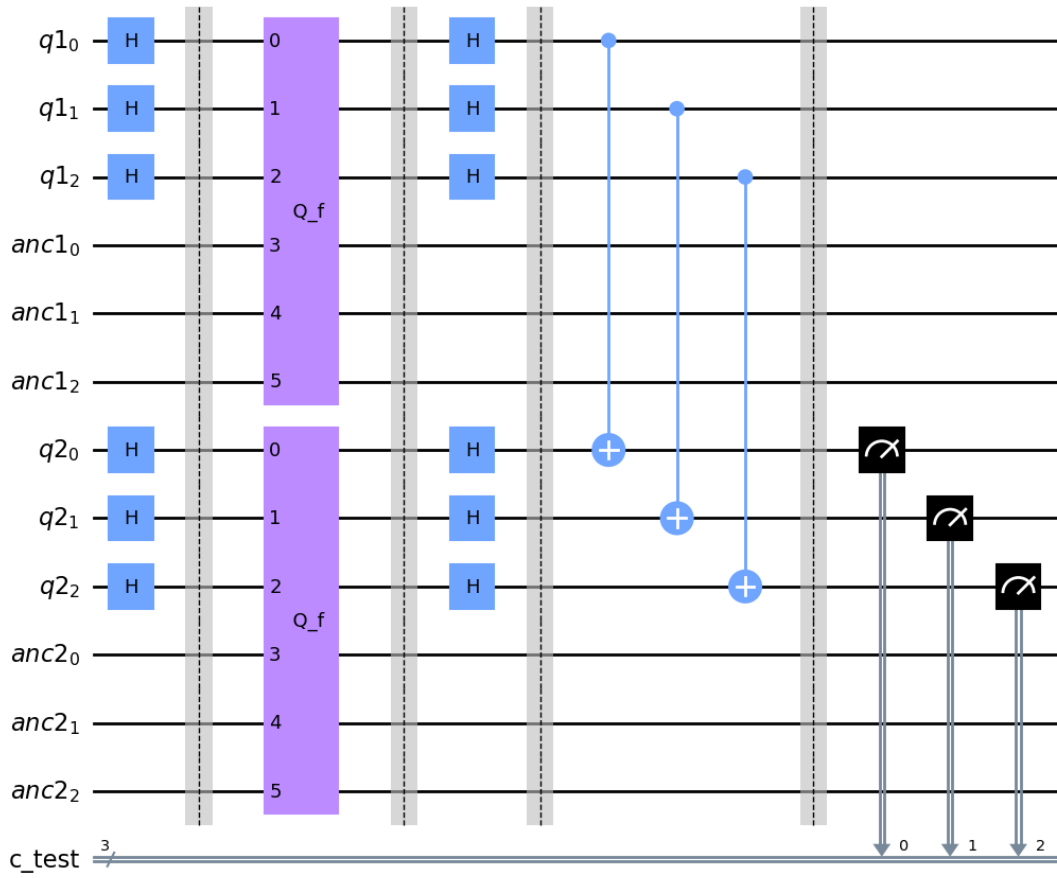
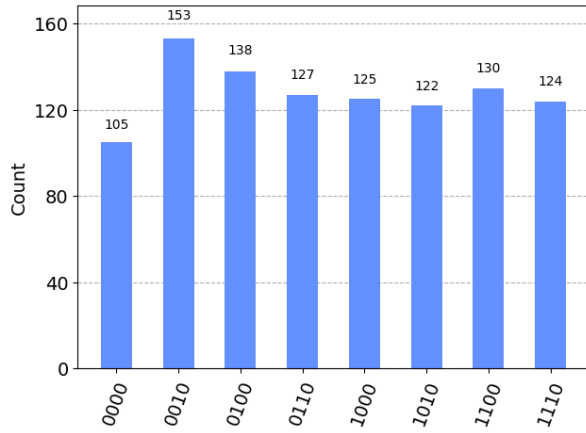
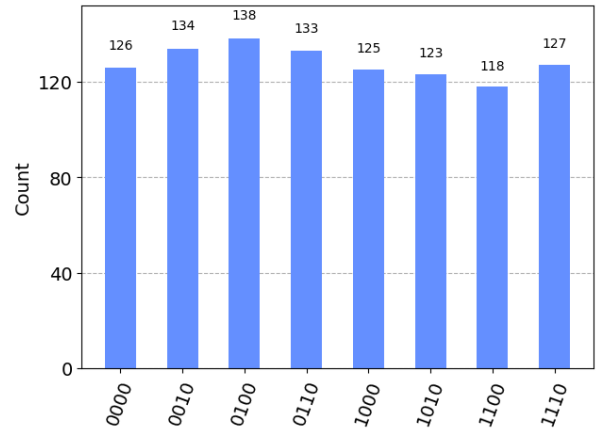


Figura 4.3: Circuito, algoritmo Simon para la regla I con  $\mathbf{c} = |001\rangle$



(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 ya presentadas las MR, así como la aplicación de MT en nuestras implementaciones de algoritmos cuánticos, siendo este el objetivo principal del trabajo, vamos a pasar a analizar todo este largo recorrido y que resultados hemos podido obtener del mismo, así como que 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 el objetivo del mismo. 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 camino 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 relación uno a uno, lo cual 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 sentadas estas bases, nos hemos dedicado a estudiar las propiedades metamórficas y como estas MR nos pueden ayudar 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 y dado que su estructura viene determinada por la definición de MT, su implementación resulta sencilla.

Por último, me gustaría destacar cómo se aprecia la necesidad y el respaldo que brindan las matemáticas para cualquier ciencia. La base principal de la creación y verificación formal de los algoritmos cuánticos es puramente matemática, al igual que la obtención de las MR de los mismos y la prueba de su validez.

Me resultó muy interesante el desarrollo realizado para la creación del algoritmo de Deutsch y cómo se puede observar que los algoritmos no salen de la nada, sino que se basan en conocimientos más abstractos como la aplicación de puertas y la prueba matemática de que estos realmente logran lo que queremos. En caso de que no se consiga el resultado deseado, el análisis que hemos realizado puede proporcionarnos una idea para 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 una mayor profundidad y número de retos, además este artículo es el que hemos utilizado para el estudio de los conceptos alrededor del *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* a una aplicación más completa 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 los algoritmos de la transformada cuántica de Fourier, en adelante QFT, y sus aplicaciones como la estimación cuántica de fase, QPE. Que se comenzaron a estudiar al igual que el algoritmo de Grover, pero no nos dio tiempo a la total comprensión y estudio de reglas metamórficas.
- Continuar con estudios de MR para otros algoritmos, como puede ser QFT o QPE, para el cual se podría utilizar la misma técnica que se utilizó al obtener la Regla III del algoritmo de BV, ya que el QPE utiliza  $QFT^{-1}$ . Además de los algoritmos de Grover o Shor.
- Estudio de otros tipos de *testing* como pruebas de mutación para relacionarlas con MT y poder aplicar ambas a la vez, como se realiza 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 obtenido, se encuentra en <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.