UNIVERSIDAD
DE GRANADA

Facultad de Ciencias

Doble grado en ingeniería informática y matemáticas

Trabajo de fin de grado

Bases de la computación e información cuánticas

Presentado por:
Daniel Antonio Gonzálvez Alfert

Tutor:
Manuel Calixto Molina

Curso académico 2022/2023

Declaración de originalidad

D./Dña. Daniel Antonio Gonzálvez Alfert

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2022-2023, es original, entendida esta, en el sentido de que no ha utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 26 de junio de 2023

Fdo: Daniel Antonio Gonzálvez Alfert

Índice general

Summary

1. What is Quantum Computing?

Quantum Computing is an emerging computational model that takes advantage of physical processes that take place in the subatomic world in order to achieve very complex calculations in a fraction of the time that the classical model would take. This new approach, unlike classical computers, which use bits to represent and process information as either 0's or 1's, uses quantum bits, or qubits, which can exist in superposition states of 0 and 1 simultaneously. These objects form the fundamental unit of information in quantum computing and they exhibit quantum phenomena such as superposition and entanglement, which are the two main properties that are harnessed in order to greatly improve the speed of calculations.

Superposition allows qubits to exist in a state that is a combination of 0 and 1. This enables quantum computers to perform computations on multiple possible states simultaneously, enabling the possibility of parallelism. Entanglement, on the other hand, establishes a correlation between qubits, meaning that the state of one qubit is linked to the state of another, regardless of the distance between them.

The way in which we represent a qubit is by expressing it by a linear combination of the vectors of a certain basis of a complex Hilbert space of dimension 2. We conventionally use what we call the computational basis, in which $|0\rangle = \begin{bmatrix}1 & 0\end{bmatrix}^T$, $|1\rangle = \begin{bmatrix}0 & 1\end{bmatrix}^T$ in Dirac's notation. Given this base, any state of the qubit can be expressed by:

$$|\psi\rangle = \mu_1 |0\rangle + \mu_2 |1\rangle ,$$

where μ_1 and μ_2 are complex numbers fulfilling $|\mu_1| + |\mu_2| = 1$. These coefficients are called amplitudes. Even though a state can be in any combination of states, once we measure it, it will change to (or *collapse* into) one of them. The squared module $|\mu_i|^2$ of the amplitud of each state of the basis that forms the state is the *probability* that we will encounter this state when measuring the qubit. Even if a qubit can be in many states, they must be kept in the dark. Therefore, being able to properly manipulate the amplitudes of different qubits in a way that results in an advantage lies at the center of quantum computing.

The manipulation of qubits is made possible through quantum logical gates, which represent operations that can be done to said qubits. The main restriction of these operations is that they must be reversible: there can only be one possible input for each output and viceversa.

2. Origins of Quantum Computing and current development

Quantum computing has its roots in the early 20th century when physicists began to unravel the fundamental principles of quantum mechanics. In the 1980s, physicist Richard Feynman and others started exploring the idea of using quantum systems for computation. Feynman proposed that quantum computers could efficiently simulate quantum systems, a task that is challenging for classical computers. However, the interest in the practical applications in this new field increased greatly when in 1994, Peter Shor discovered a quantum algorithm for factoring large numbers exponentially faster than classical algorithms. This breakthrough demonstrated the potential of quantum computers for breaking cryptographic systems based on prime factorization. The research continues nowadays. In 2019, Google's Quantum Supremacy experiment achieved a computational task that would have taken the world's fastest supercomputer thousands of years. One of the companies that lead this research is IBM, which released their roadmap for quantum development and have had great success delivering its milestones.

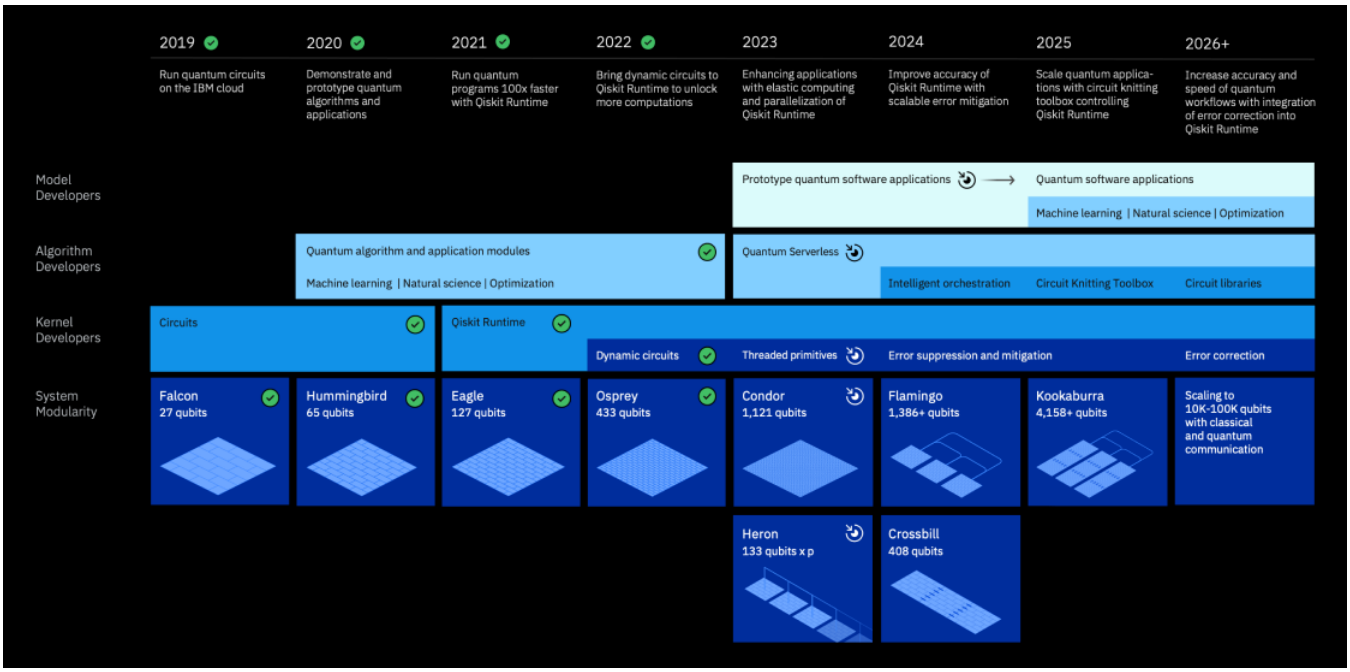


Figure 1: IBM's quantum roadmap.

3. Main objectives of this work

The porpouse of this work is to provide a comprehensive and detailed introduction to the emerging field of Quantum Computing. Starting from the needed mathematical formalism and ending in a practical implementation of quantum algorithms. This works provides an extensive initiation in fundamental concepts of quantum mechanics and quantum computing, introducing readers to the mathematical tools necessary for describing quantum systems, such as linear algebra and Dirac notation. It then delves into the principles of quantum com-

putation, covering topics such as quantum gates, quantum circuits and quantum algorithms, with a particular focus on explaining algorithms like Shor's algorithm for factoring and Grover's search algorithm.

All the objectives that were set at the start of this work have been achieved.

4. Structure of this work

Firstly, in chapter 1, we start by introducing the main mathematical concepts that we will use throughout such as the complex vector space, Kronecker product and unitary operators. Then, we go on to chapter 2, which focuses on the limitations of conventional computation, entropy and as well as introducing reversible computing, which is a very important concept given that it is necessary for quantum computing. We spend the rest of chapter 2 explaining how reversible logic gates are universal and how we can create reversible circuits to perform a certain boolean function.

Chapter 3 introduces quantum mechanics and its postulates, which will describe the behaviour of quantum systems. Is in this chapter where we also formally present the concepts of superposition and entanglement formally. We also introduce the density operator, which allows us to work with quantum systems that are not totally known. Lastly, we define the Schmidt decomposicion, which gives us a useful tool to work with entanglement.

Chapter 4 focuses on the more practical aspects of quantum computing, such as quantum logic gates and possible universal sets for them. Then, we introduce the concept of computational complexity, which we need in order to describe the complexity of quantum algorithms, which implementation we will present in the last section.

Finally, in chapter 5 consists of a brief introduction to the software tool Qiskit and a compilation of implementations of the algorithms presented in chapter 3.

Introducción

1. ¿Qué es la computación cuántica?

La computación cuántica es un modelo computacional que aprovecha los procesos físicos que tienen lugar en el mundo subatómico para realizar cálculos muy complejos en una fracción del tiempo que supondría en el modelo clásico. En este nuevo enfoque, a diferencia de los ordenadores clásicos, que utilizan bits para representar y procesar información en forma de 0 o 1, se utilizan bits cuánticos, o qubits, que pueden existir en estados de superposición de 0 y 1 simultáneamente. Estos objetos constituyen la unidad fundamental de información en la computación cuántica y exhiben fenómenos cuánticos como la superposición y el entrelazamiento, que son las dos propiedades principales que se aprovechan para mejorar enormemente la velocidad de los cálculos.

La superposición permite que los qubits existan en un estado que es una combinación de 0 y 1. Esto permite a los ordenadores cuánticos realizar cálculos en múltiples estados posibles simultáneamente, lo que posibilita el paralelismo. El entrelazamiento, por su parte, establece una correlación entre qubits, lo que significa que el estado de un qubit está ligado al estado de otro, independientemente de la distancia que los separe.

La forma en que representamos un qubit es expresándolo por una combinación lineal de los vectores de una cierta base de un espacio de Hilbert complejo de dimensión 2. Convencionalmente utilizamos lo que llamamos la base computacional, en la que $|0\rangle = \begin{bmatrix}1 & 0\end{bmatrix}^T$, $|1\rangle = \begin{bmatrix}0 & 1\end{bmatrix}^T$ en la notación de Dirac. Dada esta base, cualquier estado del qubit puede expresarse mediante:

$$|\psi\rangle = \mu_1 |0\rangle + \mu_2 |1\rangle ,$$

donde μ_1 y μ_2 son números complejos. Estos coeficientes se llaman amplitudes. Aunque un estado puede estar en cualquier combinación de estados, una vez que lo medimos, cambiará a (o *colapsará* en) uno de ellos. El módulo al cuadrado $|\mu_i|^2$ de la amplitud de cada componente de la base que forma el estado es la *probabilidad* de que nos encontremos con ese estado al medir el qubit. Por lo tanto, la capacidad para manipular adecuadamente las amplitudes de diferentes qubits de manera efectiva es fundamental en la computación cuántica.

La manipulación de los qubits es posible gracias a las puertas lógicas cuánticas, que representan las operaciones que pueden realizarse con dichos qubits. La principal restricción de estas operaciones es que deben ser reversibles: sólo puede haber una entrada posible para cada salida y viceversa.

2. Orígenes de la computación cuántica y desarrollo actual

La computación cuántica tiene sus raíces en los principios del siglo XX, cuando los físicos empezaron a descubrir los principios fundamentales de la mecánica cuántica. En los años 80, el físico Richard Feynman y otros empezaron a explorar la idea de utilizar sistemas cuánticos para la computación. Feynman propuso que los ordenadores cuánticos podrían simular eficazmente sistemas cuánticos, una tarea que supone un reto para los ordenadores clásicos. Sin embargo, el interés por las aplicaciones prácticas en este nuevo campo aumentó enormemente cuando, en 1994, Peter Shor descubrió un algoritmo cuántico para factorizar números exponencialmente más rápido que los algoritmos clásicos. Este avance demostró el potencial de los ordenadores cuánticos para romper sistemas criptográficos basados en la factorización de primos. La investigación continúa hoy en día. En 2019, el experimento Quantum Supremacy de Google logró una tarea computacional que habría llevado miles de años al superordenador más rápido del mundo. Una de las empresas que lideran esta investigación es IBM, que dio a conocer su hoja de ruta para el desarrollo cuántico y han tenido un gran éxito cumpliendo sus hitos.

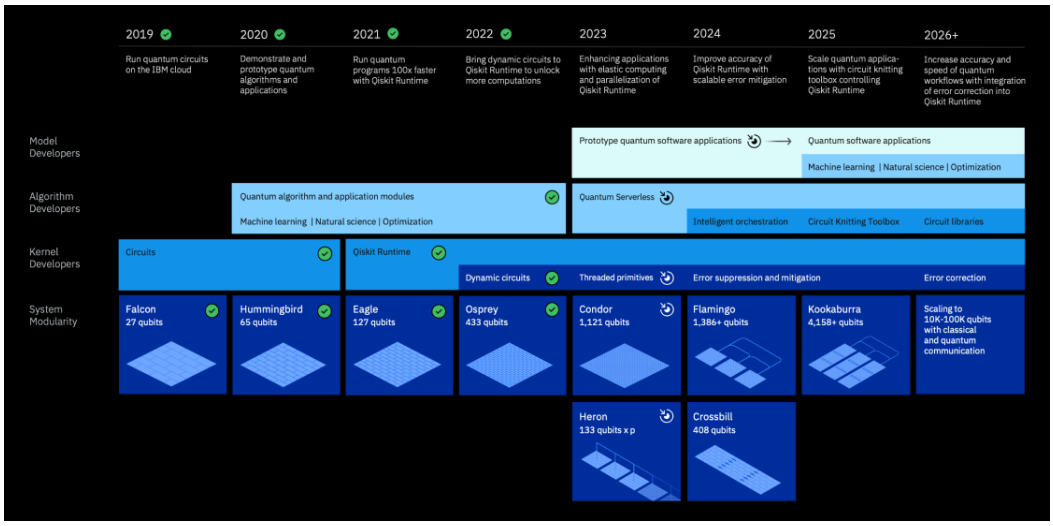


Figura 1: Hoja de ruta de IBM.

3. Objetivos principales del trabajo

El objetivo de esta obra es ofrecer una introducción completa y detallada al campo emergente de la computación cuántica. Partiendo del formalismo matemático necesario y terminando en una implementación práctica de algoritmos cuánticos. Esta obra proporciona una amplia iniciación en los conceptos fundamentales de la mecánica cuántica y la computación cuántica, introduciendo al lector en las herramientas matemáticas necesarias para describir sistemas cuánticos, como el álgebra lineal y la notación de Dirac. A continuación se adentra en los principios de la computación cuántica, abarcando conceptos como las puertas lógicas cuánticas, los circuitos cuánticos y los algoritmos cuánticos, poniendo especial atención al desarrollo teórico de algoritmos como el algoritmo de Shor para la factorización y el algoritmo

de búsqueda de Grover.

Se han alcanzado todos los objetivos fijados al inicio de este trabajo.

4. Estructura del trabajo

Al principio del trabajo, en el capítulo 1, comenzamos introduciendo los principales conceptos matemáticos que utilizaremos a lo largo del mismo como el espacio vectorial complejo, el producto de Kronecker y los operadores unitarios. A continuación, pasamos al capítulo 2, que se centra en las limitaciones de la computación convencional, la entropía y la introducción de la computación reversible, que es un concepto muy importante, ya que sirve de antesala para la computación cuántica. Pasamos el resto del capítulo explicando cómo las puertas lógicas reversibles son universales y cómo podemos crear circuitos reversibles para implementar una determinada función booleana.

El capítulo 3 introduce la mecánica cuántica y sus postulados, que definirán el comportamiento de los sistemas cuánticos. Es en este capítulo donde también presentamos formalmente los conceptos de superposición y entrelazamiento. También introducimos el operador de densidad, que nos permite trabajar con sistemas cuánticos que no son totalmente conocidos. Por último, definimos la descomposición de Schmidt, que nos proporciona una herramienta útil para trabajar con el entrelazamiento.

El capítulo 4 se centra en los aspectos más prácticos de la computación cuántica, como las puertas lógicas cuánticas y los posibles conjuntos universales para ellas. A continuación, introducimos el concepto de complejidad computacional, que necesitamos para describir la complejidad de los algoritmos cuánticos, cuya implementación presentaremos en la última sección.

El último capítulo de este trabajo es el 5, que consiste en una breve introducción a la herramienta de software Qiskit y una recopilación de implementaciones de los algoritmos presentados en el capítulo 3.

5. Fuentes bibliográficas principales

En el capítulo 2 se usan principalmente las fuentes [FHA98] [BAP+12] y [Tah16]. En los capítulos 3 y 4 se usa casi exclusivamente [NC10].

1 Primeras definiciones

1.1. El espacio vectorial complejo

Los posibles estados puros de un sistema físico cuántico se identifican con los vectores de un espacio de Hilbert complejo. Nos centramos en el caso finito-dimensional, en el cual utilizaremos el espacio vectorial complejo $\mathbb{C}^n$ (usualmente con $n = 2^m, m \in \mathbb{N}$) para modelar distintos tipos de sistemas físicos, como el formado por un átomo con n niveles de energía distintos, o $\log_2 n$ átomos con dos niveles. Para llevar a cabo dicha representación, utilizaremos objetos cuánticos denominados qubits, que se explicarán con detalle más adelante.

El estado de los sistemas mencionados vendrá dado por un vector columna complejo. Nosotros utilizaremos lo que se denomina *notación de Dirac* o *notación bra-ket*. En ella identificaremos los vectores complejos con los denominados *kets*:

$$|a\rangle = \left[a_1, a_2, \ldots, a_n\right]^T.$$

Este espacio vectorial cuenta con la suma y producto por escalares usual. Sean $|a\rangle, |b\rangle \in \mathbb{C}^n$, $c \in \mathbb{C}$:

$$|a\rangle + |b\rangle = \left[a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n\right]^T \qquad c\,|a\rangle = \left[ca_1, ca_2, \ldots, ca_n\right]^T.$$

Por otra parte, es posible dotar a este espacio vectorial con un producto escalar. Usaremos el producto escalar usual definido como:

$$\langle a, b\rangle := \sum_{i=1}^{n} \overline{a_i} b_i.$$

Siendo $\overline{a_i}$ la conjugación compleja de a_i. El producto escalar nos permite además definir una norma para cualquier vector en $\mathbb{C}^n$. De modo que dado $a \in \mathbb{C}^n$, definimos la norma de a como:

$$\|a\| = \sqrt{\langle a, a\rangle}.$$

$\mathbb{C}^n$ con la norma que acabamos de definir es, en particular, un *espacio de Hilbert*. Por otro lado, al haber definido el producto escalar, es posible hacer uso del concepto de ortogonalidad. Se dice que dos vectores $a, b \in \mathbb{C}^n$ son ortogonales si $\langle a, b\rangle = 0$, en cuyo caso denotaremos $a \perp b$. Una vez que hemos definido ortogonalidad y tenemos una norma, podemos presentar el concepto de base ortonormal.

Definición 1.1. Sea una base $B = \{v_1, v_2 \dots v_n\}$ de $\mathbb{C}^n$. Esta base se dice *ortonormal* si para $i \neq j$ con $i, j = 1, 2 \dots n, v_i \perp v_j$ y además $||v_i|| = 1$ con $i = 1 \dots n$.

Una vez que ya hemos definido un espacio y la base que utilizamos para la representación de sus vectores, pasamos a otro concepto que nos será útil para entender la notación que se presentará más adelante: el espacio vectorial dual.

Definición 1.2. Sean dos espacios vectoriales V, W, ambos sobre un cuerpo $\mathbb{K}$. Un *operador* lineal es una aplicación $\phi : V \to W$ cumpliendo, para cada $a, b \in V, c \in \mathbb{K}$:

$$\phi(a + cb) = \phi(a) + c\phi(b).$$

Definición 1.3. Sea el espacio vectorial $\mathbb{C}^n$ definimos su *espacio dual*, $(\mathbb{C}^n)^*$, como el conjunto de operadores lineales $\phi : \mathbb{C}^n \to \mathbb{C}$.

Los operadores lineales pertenecientes a un espacio dual se denominan, en general, $funcionales$. Además, sabemos que los funcionales de un espacio dual tienen una forma determinada. Esta forma viene dada por el *Teorema de Riesz*.

Teorema 1.1 (Teorema de representación de Riesz)**.** *Fijamos $x \in \mathbb{C}^n$ y definimos el funcional:*

$$\psi_x(y) = \langle y, x \rangle,$$

que pertenece a $(C^n)^*$. El teorema de representación de Riesz nos dice que la función:

$$\phi : (C^n) \to (C^n)^*$$
$$\phi(x) = \psi_x,$$

es un anti-isomorfismo isométrico, es decir, siendo $\lambda \in \mathbb{C}$ y * la conjugación compleja:

$$\phi(\lambda x) = \lambda^* \phi(x)$$
$$\|x\| = \|\phi(x)\|$$

Este teorema nos permite construir la contraparte dual de cualquier vector de $\mathbb{C}^n$, que ahora sabemos que es única. Sea un vector $v \in \mathbb{C}^n$, escrito como:

$$v = \left[v_1, \dots, v_n\right]^T.$$

Definimos su contraparte dual $v^* \in (\mathbb{C}^n)^*$ como el funcional:

$$v^* : \mathbb{C}^n \to \mathbb{C}$$
$$v^*(x_1, \dots, x_n) = \sum_{i=0}^{n} \overline{v_i} x_i.$$

Llamaremos *bra* a estas construcciones: si $|v\rangle \in \mathbb{C}^n$, entonces $\langle v| := |v\rangle *$. Los bra pueden entenderse como vectores fila. Aprovechando esta identificación, podemos ver la clara correlación entre la notación utilizada y el producto escalar:

$$\langle a|b\rangle = a^*(b) = \langle a, b\rangle = \sum_{i=1}^{n} \overline{a_i} b_i = \left[\overline{a_1}, \ldots, \overline{a_n}\right] \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}.$$

A la vez, podemos cambiar el orden del producto y obtener una matriz:

$$|b\rangle \langle a| = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \left[\overline{a_1}, \ldots, \overline{a_n}\right] = \begin{bmatrix} b_1\overline{a_1} & \ldots & b_1\overline{a_n} \\ \vdots & & \vdots \\ b_n\overline{a_1} & \ldots & b_n\overline{a_n} \end{bmatrix}.$$

Las matrices definidas de esta manera tienen propiedades que pueden inferirse gracias a las facilidades que nos provee esta notación. Por ejemplo, la matriz $|a\rangle \langle b|$ que acabamos de presentar representa un operador lineal con la siguiente propiedad:

La matriz $|a\rangle \langle b|$ aplicada a $|c\rangle \in \mathbb{C}^n$ se escribe como:

$$|a\rangle \langle b| (|c\rangle) = |a\rangle \langle b|c\rangle = \langle b|c\rangle |a\rangle .$$

Que no es más que el vector (o ket) $|a\rangle$ multiplicado por el número complejo $\langle b|c\rangle$. Por otro lado, es un interesante notar que las matrices descritas como $A = |a\rangle \langle a|$ siempre son operadores *semidefinidos positivos*. Esto es fácil de ver a partir de la propia notación:

$$\langle b| A |b\rangle = \langle b|a\rangle \langle b|a\rangle = \langle b|a\rangle^2 \geq 0.$$

1.2. Producto de Kronecker

Empezaremos definiendo, de forma general, el concepto de *producto tensorial*, para, a partir de ahí, presentar una de las herramientas más importantes en nuestro arsenal a la hora de abordar problemas que necesiten de la representación de más información: el *producto de Kronecker*. Esta es la formalización matemática que utilizaremos para, dicho informalmente, añadir más dimensiones con las que trabajar, o visto de otra manera, juntar distintos espacios vectorial para acabar con uno más grande.

Físicamente, podemos pensar en el producto tensor $\mathbb{C}^n \otimes \overset{m}{\ldots} \otimes \mathbb{C}^n = \mathbb{C}^{n^m}$ como representando sistemas de m partículas o átomos de n niveles. En primer lugar necesitaremos una primera definición indispensable:

Definición 1.4. Sean tres espacios vectoriales, U , V y W todos sobre el mismo cuerpo $\mathbb{K}$. Se dice que una aplicación B que asocia una tupla $(u, v) \in U \times$ V a un vector $w \in W$ es *bilineal* cuando es lineal en cada componente.

Sean $\lambda \in \mathbb{K}$, $u_1, u_2 \in U$, $v_1, v_2 \in V$.

B: U × V → W es bilineal si:

$$B(\lambda u_1 + u_2, v_1) = \lambda B(u_1, v_1) + B(u_2, v_1)$$
$$B(u_1, \lambda v_1 + v_2) = \lambda B(u_1, v_1) + B(u_1, v_2)$$

Definición 1.5. Sean $\mathbb{C}^n, \mathbb{C}^m$ y $Z(\mathbb{C})$ espacios vectoriales complejos. Se define el *producto tensorial* como un nuevo espacio vectorial denotado por $\mathbb{C}^n \otimes \mathbb{C}^m$ junto con un operador bilineal $\otimes$:$\mathbb{C}^{nm} \to \mathbb{C}^n \otimes \mathbb{C}^m$ de modo que para cada operador bilineal
$h : \mathbb{C}^{nm} \to Z$ existe una única biyección lineal $\bar{h}$: $\mathbb{C}^n \otimes \mathbb{C}^m$ tal que $h = \bar{h} \circ \otimes$.

Dicho de otra forma mucho más coloquial, la definición hace referencia a una operación entre espacios vectoriales, dando como resultado un espacio vectorial mayor en el cual se contenga toda la información de los dos anteriores. A continuación enunciaremos algunas propiedades del producto tensorial.

Proposición 1.1. *Sean dos espacios vectoriales U,V y W. El producto tensorial cumple las siguientes propiedades.*

1. *Si U y V tienen dimensiones n,m* $\in \mathbb{N}$ *respectivamente, entonces la dimensión de U×V es nm.*
2. $(U \otimes V) \otimes W \cong U \otimes (V \otimes W)$.
3. $(U \otimes V) \cong (V \otimes U)$.

Finalmente, podemos presentar una particularización del producto tensorial denominado *producto de Kronecker*. Este producto viene de definir la función $\otimes$ de la definición de producto tensorial como:

Dados $n, m, p, q \in \mathbb{N}$, A $\in \mathbb{C}^{nm}$, B $\in \mathbb{C}^{pq}$:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{p1} & \cdots & b_{pq} \end{bmatrix}$$

$$\otimes : \mathbb{C}^{nm} \times \mathbb{C}^{pq} \to \mathbb{C}^{pnqm}$$
$$(A, B) \longmapsto A \otimes B$$

$$A \otimes B = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & \cdots & a_{11}b_{1q} & \cdots & \cdots & a_{1n}b_{11} & a_{1n}b_{12} & \cdots & a_{1n}b_{1q} \\ a_{11}b_{21} & a_{11}b_{22} & \cdots & a_{11}b_{2q} & \cdots & \cdots & a_{1n}b_{21} & a_{1n}b_{22} & \cdots & a_{1n}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{11}b_{p1} & a_{11}b_{p2} & \cdots & a_{11}b_{pq} & \cdots & \cdots & a_{1n}b_{p1} & a_{1n}b_{p2} & \cdots & a_{1n}b_{pq} \\ \vdots & \vdots & & \vdots & \ddots & & \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & & \ddots & \vdots & \vdots & & \vdots \\ a_{m1}b_{11} & a_{m1}b_{12} & \cdots & a_{m1}b_{1q} & \cdots & \cdots & a_{mn}b_{11} & a_{mn}b_{12} & \cdots & a_{mn}b_{1q} \\ a_{m1}b_{21} & a_{m1}b_{22} & \cdots & a_{m1}b_{2q} & \cdots & \cdots & a_{mn}b_{21} & a_{mn}b_{22} & \cdots & a_{mn}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{p1} & a_{m1}b_{p2} & \cdots & a_{m1}b_{pq} & \cdots & \cdots & a_{mn}b_{p1} & a_{mn}b_{p2} & \cdots & a_{mn}b_{pq} \end{bmatrix}$$

O lo que es equivalente, con una pequeña sobrecarga de notación:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}$$

Nóte que n,m,p o q pueden ser uno, de manera que también puede aplicarse a vectores tanto fila como columna y sus combinaciones con matrices. A nosotros nos resultará de especial interés el producto de Kronecker de vectores fila con otros vectores fila y el de matrices con otras matrices.

En general, utilizaremos una pequeña sobrecarga de notación cuando trabajemos kets. Siempre y cuando no genere confusión, omitiremos el símbolo $\otimes$ o incluso la notación del propio ket, de manera que, dados $|v\rangle, |w\rangle \in C^n$, $|v\rangle \otimes |w\rangle = |v\rangle |w\rangle = |vw\rangle$.

En particular, podemos aplicar el producto de Kronecker a una matriz o vector consigo mismo. Utilizaremos una notación especial para este caso. Siendo $|v\rangle \in \mathbb{C}^n$ y $k \in \mathbb{N}$ definimos:

$$|v\rangle^{\otimes k} := |v\rangle \otimes \ldots k \ldots \otimes |v\rangle .$$

Ejemplo 1.1. Una matriz que aparecerá a menudo en este trabajo, y que se usa para crear superposición de estados de la base computacional, es la matriz de Hadamard, que tiene la forma:

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

Sin embargo, en muchas ocasiones que serán presentadas más adelante, será necesario hacer el producto de Kronecker de esta matriz consigo misma. Por suerte, en este caso contamos con una fórmula general [NC10][p. 75]. Sean $n, k \in \mathbb{N}$ $B_{ket} = \{|1\rangle .. |n\rangle\}$ una base de $\mathbb{C}^n$, y $B_{bra} = \{\langle 1| .. \langle n|\}$ el conjunto de los bras asociados a cada uno de los kets de la base. Entonces:

$$H^{\otimes k} = \frac{1}{\sqrt{2^k}} \sum_{x \in B_{ket}} \sum_{y \in B_{bra}} (-1)^{\langle x|y\rangle} |x\rangle \langle y| .$$

Afortunadamente, contamos con la fórmula general para esta puerta tan importante, pero en general no contaremos con dicha fórmula para el desarrollo del producto tensorial de una matriz cualquiera.

1.3. Operadores

Un operador nos es más que otro nombre para una aplicación entre espacios vectoriales. En nuestro caso nos resultarán de interés aquellos que son lineales. Estas aplicaciones se

expresan como matrices, fijada una base, de modo que aplicar el operador a un vector no es más que multiplicarlo por su matriz asociada.
En el caso de que ambos espacios tengan dimensión finita, las dimensiones de la matriz, $n \times m$, coinciden con las dimensiones del espacio de entrada y el de salida.

Para saber como actúa un operador sobre los vectores de un espacio vectorial, basta saber la imagen de una base del mismo.

Sean espacios vectoriales U,V ambos sobre el mismo cuerpo $\mathbb{K}$. Sean $B_v = v_1, \ldots, v_n$ y $B_u = u_1, \ldots, u_n$ bases de U y V, respectivamente. Sea A: U$\rightarrow$V un operador. La representación matricial de A es:

$$M(A, B_v \leftarrow B_u) = \left[(Au_1)_{B_v} \ldots (Au_n)_{B_v}\right].$$

Gracias a la matriz asociada al operador, se nos abren las puertas para aprovechar las propiedades de las matrices y obtener información relevante sobre él. Empezaremos aprovechando la teoría de diagonalización. A partir de ahora nos referiremos a un operador indistintamente por su aplicación y su matriz asociada en la base usual, o también llamada computacional.

Definición 1.6. Sea V un espacio vectorial sobre un cuerpo $\mathbb{K}$ y A un operador lineal en V. Se dice que λ es un *autovalor* de la matriz asociada a A $\in \mathbb{K}$ si, dado $|v\rangle \in$ V:

$$A|v\rangle = \lambda|v\rangle.$$

En este caso, se dice que $|v\rangle$ es un *autovector* asociado al autovalor λ. El subespacio vectorial generado por un autovector se denomina *autoespacio*. El conjunto de autovalores de un operador lineal se llama *espectro*.

Proposición 1.2. *Dado un operador A y su matriz asociada M_A. Los autovalores de M_A son las raíces del polinomio dado por:*

$$det(M - \lambda I_V)$$

Este polinomio se denomina polinomio característico de la matriz.

Otra cualidad interesante de los autovalores de una matriz, es que la suma de los mismos es igual a la traza de la matriz, de modo que si $M = (m_{ij})$ con $i, j = 1, \ldots, n$:

$$tr(M) = \sum_i^n m_{ii} = \sum_i \lambda_i$$

Por otro lado, aprovechando que acabamos de definir la traza de una matriz cuadrada, presentamos una identidad que será muy útil. Si $\||\psi\rangle\| = 1$ y A es una matriz, entonces se cumple que:

$$tr(A|\psi\rangle\langle\psi|) = \langle\psi|A|\psi\rangle. \tag{1.1}$$

Definición 1.7. Una matriz M se dice *diagonalizable* si es semejante a una matriz diagonal. En

otras palabras, si existen matrices regulares P y D, siendo D diagonal, tal que:

$$M = PDP^{-1}$$

Esto es equivalente a decir que existe una base formada por autovectores de M y dicha matriz expresada en esta base es una matriz diagonal. Además, la matriz resultante tiene por diagonal los autovalores de M, apareciendo un número de veces igual a su multiplicidad como raíces.

Definición 1.8. Una matriz M $\in \mathbb{R}^{n\times n}$ se dice *ortogonal* si cumple que $M^{-1} = M^T$.

Proposición 1.3. *Sea M una matriz diagonalizable. Si la base B formada por los autovectores de M es una base ortonormal, entonces la matriz P de cambio de base es ortogonal y, por lo tanto, PMP^T es una matriz diagonal.*

Demostración. Sea $M \in \mathbb{R}^{n\times n}$ una matriz diagonalizable. La matriz P^{-1}, que renombraremos como Q se construye como los vectores columna de la base ortonormal formada por autovectores. En ese caso:

$$QQ^T = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{nn} \end{bmatrix} = I_{n\times n}$$

Esto se debe a que al realizar la multiplicación de la columna i por la columna j, si i = j, nos da 1 al ser esta multiplicación análoga al producto interno en $\mathbb{R}$, y en otro caso, nos da 0 por ser los vectores de la base ortogonales 2 a 2. Por ende

$$QQ^T = I_{n\times n} \implies Q^T = Q^{-1} \implies Q = (Q^{-1})^T \implies P^{-1} = P^T.$$

Y si P^{-1} es ortogonal, claramente P también lo es. □

Esta proposición tendrá su análogo en el espacio vectorial complejo, pero para ello deberemos introducir algunos conceptos previos.

Definición 1.9. Dada una matriz M $\in \mathbb{C}^{n\times m}$ se define su *adjunta* como:

$$M^\dagger = (M^*)^T.$$

Siendo M^* la operación de conjugación compleja elemento a elemento y T la transposición. Además, si $M^\dagger = M$, se dice que M es *Hermitiana*.

Esta operación cumple que si A es un operador lineal en un espacio de Hilbert V, $A^\dagger$ es el único operador lineal tal que para todo vector $|v\rangle, |w\rangle \in V$:

$$\langle v| A |w\rangle = \left\langle A^\dagger\, v \middle| w \right\rangle$$

Ejemplo 1.2. La adjunta de un vector cualquiera es su vector dual asociado. Es decir, siendo $v \in \mathbb{C}^n$, $|v\rangle^\dagger := \langle v|$ y viceversa. Esto se debe a que claramente $(|v\rangle^\dagger)^\dagger = |v\rangle$.

Además, esta operación es lineal. Por tanto, si tenemos una combinación lineal escrita como $|v\rangle = a_1 |v_1\rangle + a_2 |v_2\rangle$, entonces $\langle v| = a_1^* \langle v_1| + a_2^* \langle v_2|$.

Ejemplo 1.3. Un caso particular de matriz Hermitiana son los *proyectores*[NC10][p. 70].Supongamos un espacio vectorial V de dimensión n, y un subespacio W con dimensión $m < n$. Siempre podemos escoger una base de V $B_V = |1\rangle \dots |n\rangle$ ortonormal y cumpla que el subconjunto $B_W = |1\rangle \dots |m\rangle$ sea una base de W. Esto se debe a que siempre se puede conseguir una base de W, que se complete con vectores linealmente independientes hasta conseguir una base de V, para después aplicar el algoritmo de Grand-Schmidt, que permite transformar cualquier base en una ortonormal. Podemos realizar la siguiente construcción:

$$P_W := \sum_{i=1}^{m} |i\rangle \langle i|$$

La matriz P resultante se denomina proyector y es Hermitiana. Esto puede demostrarse utilizando que para cualquier vector $v \in \mathbb{C}^n$, $(|v\rangle \langle v|)^\dagger = |v\rangle \langle v|$. Esto puede verse claramente utilizando que $(|v\rangle)^\dagger = \langle v|$ y $(\langle v|)^\dagger = |v\rangle$. Por ende:

$$P_W^\dagger = (\sum_{i=1}^{m} |i\rangle \langle i|)^\dagger = \sum_{i=1}^{m} (|i\rangle \langle i|)^\dagger = \sum_{i=1}^{m} (|i\rangle \langle i|).$$

Además, esta matriz es la representación de la proyección de los vectores de V en el subespacio vectorial W. Esto puede demostrarse fácilmente tomando ventaja de la propia notación.

Dado $|v\rangle \in C^n$:

$$|v\rangle = \sum_{i=1}^{n} \lambda_i |i\rangle \, , P |v\rangle = \sum_{i=1}^{m} \lambda_i \sum_{j=1}^{n} |i\rangle \langle i|j\rangle \, .$$

Sin embargo, si $i \neq j \, \langle i|i\rangle = 0$ debido a la ortogonalidad de los vectores de la base mientras que si $i = j \, \langle i|j\rangle = 1$, dado que también son unitarios. Por lo tanto:

$$P |v\rangle = \sum_{i=1}^{m} \lambda_i |i\rangle \, .$$

Que es justamente la forma de los vectores de W.

Por otra parte, podemos obtener lo que se denomina *complemento ortogonal* de P, que es la proyección a $V - W$. Esta puede calcularse como la matriz $I - P$. La demostración de este hecho es sencilla. Sea $|v\rangle \in V$

$$|v\rangle = \sum_{i=1}^{n} \lambda_i |i\rangle$$

$$\sum_{i=1}^{n} |i\rangle \langle i| \, v = \sum_{i=1}^{n} |i\rangle \langle i| \sum_{j=1}^{n} \lambda_j |j\rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_j |i\rangle \langle i|j\rangle = \sum_{i=1}^{n} \lambda_i |i\rangle = |v\rangle \, .$$

Por tanto:

$$I = \sum_{i=1}^{n} |i\rangle \langle i| \implies I - P = \sum_{i=m}^{n} |i\rangle \langle i| \, .$$

Lo cual claramente es una construcción análoga a P, pero con los últimos $n - m$ vectores de la base de V, los cuales forman una base de $V - W$, produciendo así un proyector a ese subespacio.

1.4. Operadores normales y unitarios

Los operadores normales, y mayoritariamente aquellos unitarios, serán la forma fundamental en la que representaremos transformaciones que ocurren en un sistema cuántico.

Definición 1.10. Sea un operador A y su matriz $M_A \in \mathbb{C}^{n\times n}$. Se dice que A es *normal* cuando su matriz asociada conmuta con su matriz adjunta:

$$M_A M_A^\dagger = M_A^\dagger M_A.$$

Un caso particular de las matrices normales es el de las matrices unitarias.

Definición 1.11. Sea un operador A y su matriz $M_A \in \mathbb{C}^{n\times n}$. Se dice que A es *unitario* cuando su matriz inversa es su matriz adjunta:

$$M_A M_A^\dagger = M_A^\dagger M_A = I_{n\times n}.$$

Ahora que contamos con esta definición, podemos actualizar la proposición 2.2 para un espacio vectorial complejo.

Proposición 1.4. *Sea M una matriz diagonalizable. Si la base B formada por los autovectores de M es una base ortonormal, entonces la matriz P de cambio de base es unitaria y, por lo tanto, $PMP^\dagger$ es una matriz diagonal.*

Demostración. Utilizamos exactamente el mismo argumento que en la proposición previamente mencionada.

$$QQ^\dagger = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} \overline{a_{11}} & \cdots & \overline{a_{n1}} \\ \vdots & \ddots & \vdots \\ \overline{a_{1n}} & \cdots & \overline{a_{nn}} \end{bmatrix} = I_{n\times n}$$

Esto se debe a que al realizar la multiplicación de la columna i por la columna j, si i = j, nos da 1 al ser esta multiplicación análoga al producto interno en $\mathbb{C}$, y en otro caso, nos da 0 por ser los vectores de la base ortogonales 2 a 2. Por ende

$$QQ^\dagger = I_{n\times n} \implies Q^\dagger = Q^{-1} \implies Q = (Q^{-1})^\dagger \implies P^{-1} = P^\dagger.$$

Y si P^{-1} es unitaria, claramente P también lo es. □

A continuación presentaremos uno de los teoremas más importantes que utilizaremos, ya que nos permitirá caracterizar las matrices normales que se nos presenten.

Teorema 1.2 (Descomposición espectral)**.** *Sea un espacio vectorial complejo V de dimensión finita y $M : V \to V$ un operador. M es diagonalizable si y solo si es normal [NC10][p. 72].*

Por otro lado, tenemos una propiedad más débil que la diagonalización, pero que nos aporta operatividad más adelante.

Proposición 1.5. *Sea A una matriz cuadrada. Entonces existen matrices unitarias U y V, así como D una matriz diagonal de entradas no negativas tales que:*

$$A = UDV.$$

Corolario 1.1. *Una matriz normal M, con autovalores $\lambda_1, \ldots, \lambda_n$ y autovectores $|1\rangle, \ldots, |n\rangle$ puede descomponerse siguiendo la fórmula:*

$$M = \sum_{i=1}^{n} \lambda_i |i\rangle \langle i|$$

Esta construcción se denomina descomposición espectral de M.

Ejemplo 1.4. Una matriz semidefinida positiva es hermitiana ,por lo que siempre es diagonalizable y por ende tiene una descomposición espectral.

El corolario anterior nos permite definir un nuevo tipo de operadores: los operadores función. Estos son, a todos rasgos, operadores dependientes funcionalmente de un parámetro.

Definición 1.12. Sea M una matriz normal y

$$M = \sum_{i=1}^{n} \lambda_i |i\rangle \langle i|,$$

su descomposición espectral. Por otro lado, tomamos una función $f : \mathbb{C} \to \mathbb{C}$ cualquiera tal que esté, como mínimo, definida sobre los autovalores de M. Definimos el *operador función* como:

$$f(M) = \sum_{i=1}^{n} f(\lambda_i) |i\rangle \langle i|$$

Ejemplo 1.5. Sea la matriz

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = 1 |0\rangle \langle 0| - 1 |1\rangle \langle 1|$$

Siendo 1 y -1 los autovalores y la base $\{|0\rangle, |1\rangle\} = \{[1,0]^T, [0,1]^T\}$. Tomamos $f(z) = exp(\theta z)$. De modo que f(Z) puede definirse como:

$$f(Z) = exp(\theta Z) = f(1) |0\rangle \langle 0| - f(1) |1\rangle \langle 1| = exp(\theta) |0\rangle \langle 0| + exp(-\theta) |1\rangle \langle 1| = \begin{bmatrix} e^{\theta} & 0 \\ 0 & e^{-\theta} \end{bmatrix}$$

Otras dos definiciones que serán necesarias en el contexto de los operadores serán la de *conmutador* y *anticonmutador*.

Definición 1.13. El *conmutador* de dos operadores A y B se define como:

$$[A, B] := AB - BA.$$

Mientras que el *anticonmutador* se define como:

$$\{A, B\} := AB + BA.$$

El conmutador de dos operadores nos da una medida de como de conmutables son, de ahí su nombre. El anti conmutador tiene la misma función para la anti conmutación (dos matrices anti conmutan si $AB = -BA$).

2 Fundamentos de la computación reversible

2.1. Introducción

La unidad de información más fundamental que podemos tomar es el bit, que puede usarse para representar un estado binario. Por ejemplo, 1 como respuesta afirmativa a una pregunta y 0 para una respuesta negativa. Todos los ordenadores que conocemos usan esta base para tratar la información que utilizan en sus diversas actividades. Los ordenadores son capaces transformar esta información mediante unos dispositivos electrónicos llamados transistores. El funcionamiento de estos componentes es relativamente sencillo: permiten o imposibilitan el paso por un canal de una pequeña corriente eléctrica dependiendo de otra. De este modo, usamos el estado binario del transistor como representación física de un modelo lógico abstracto como son los bits.

El 16 de abril de 1965, el cofundador de la famosísima empresa de ingeniería Intel, Gordon E. Moore, enunció lo que sería conocido como la *Ley de Moore*. Esta ley empírica dictamina que el número de transistores de un microprocesador se doblaría aproximadamente cada dos años. Aun pudiendo sonar fantasiosa, la predicción del creador del gigante tecnológico se mantuvo vigente a lo largo de décadas, sobreviviéndolo a él mismo, tras su muerte el 24 de marzo de 2023.

La intuición nos dice que para que haya más de algo en el mismo espacio, este algo debe disminuir en tamaño. Siguiendo esta regla, los transistores han ido disminuyendo de tamaño con el tiempo, pasando de un tamaño de 10μm en 1971, a 3nm en 2022: más de 300 veces más pequeño. Sin embargo, este sueño de crecimiento sin cotas parece estar encontrándose con diversos obstáculos, como el coste eléctrico o la eficiencia, que comentaremos más adelante.

La computación reversible es un nuevo paradigma de la computación que nos ayudará a sortear los obstáculos principales de la computación tradicional. La computación reversible es un modelo de computación que se diferencia de la computación clásica en que todas sus operaciones son invertibles, es decir, que pueden deshacerse por completo sin pérdida de información. Esta propiedad es fundamental para disciplinas incipientes como la computación cuántica, debido a que la evolución temporal del estado de un sistema físico debe preservar su norma, que más tarde veremos que implica conservación de la información. Sin embargo, la computación reversible también es importante en la computación clásica, ya que permite una mayor eficiencia en la gestión de la información y una reducción en el consumo de energía previa al borrado de información. En la computación clásica, muchas operaciones comunes, como la eliminación de bits o la suma de números binarios, no son reversibles y, por lo tanto, pueden provocar la pérdida de información. Como consecuencia, la computación reversible permite una mejora en la eficiencia energética y una gestión de la información más precisa.

2.2. Termodinámica y restricciones de la computación convencional.

2.2.1. Coste energético

La segunda ley de la termodinámica nos dice que en un sistema termodinámico aislado, cualquier proceso espontáneo siempre tenderá al aumento de entropía de dicho sistema. La entropía, en el sentido estadístico de la misma, es una magnitud física que mide la probabilidad de encontrar un estado determinado. [FHA98][p. 142]

Ejemplo 2.1. Por ejemplo, imaginemos un número N de monedas con las caras marcadas por las letras A y B y perfectamente equilibradas. La probabilidad de obtener A o B es 0.5. Si tirásemos todas esas monedas, el sistema resultante podría, con probabilidad no nula, resultar en todas las caras teniendo una misma letra determinada, independientemente del número N de monedas. Sin embargo, sabiendo que este tipo de sistemas siguen una distribución de Bernuilli con $p = 0.5$, es fácil calcular dicha probabilidad, que resulta en $\frac{1}{2^N}$. Este estado del sistema es muy poco probable comparado con casi cualquier otro por lo que se dice este estado tiene una *baja entropía*.

En contraste, podríamos considerar la configuración del sistema definido como tener M monedas con la cara A, y $M - N$ con la cara B. Intuitivamente es mucho más probable encontrar este estado, ya que su probabilidad es de $\frac{2^N-2}{2^N}$ que es aproximadamente 1. Por ende, se dice que este estado tiene una *alta entropía*.

A continuación, consideremos el mismo caso que en el ejemplo anterior, pero en vez de usar monedas, utilizamos N bits, cada uno en un estado aleatorio. Imaginemos que ahora cogemos alguno de ellos y realizamos un *borrado de información*, es decir, ponemos el transistor en un estado A $\in 0, 1$ prefijado. La probabilidad de tener una configuración de bits con un bit fijado es menor que la de encontrar una sin ninguno fijado. De este modo, la entropía ha disminuido. Sin embargo, la entropía no puede disminuir, por lo que el ambiente tiene que ganar entropía en forma de calor. Esto se debe a que se debe compensar por la pérdida de información: no sabemos, sin haber medido, cuál era el estado del transistor antes de cambiarlo a A. Ahora, cabe preguntarse en qué proporción esta energía se disipa. El principio de Landauer responde a esta pregunta. Rolf Landauer propuso en 1961 un principio que nos permite saber la energía mínima que se disipa al borrar un bit de información[BAP+12][p. 1].

Teorema 2.1. *El principio de Landauer nos dice que la energía que se disipa al borrar un bit de información es:*

$$E = k_B T \ln 2,$$

siendo E la energía disipada en Julios, T la temperatura ambiente en Kelvins y k_B la constante de Boltzmann, que tiene un valor de $1.38 \cdot 10^{-23} \frac{J}{K}$.

Los ordenadores actuales operan en torno a un millón de veces este límite, pero se cree que en las próximas décadas nos acercaremos "peligrosamente" a él. Esto genera un grave problema, puesto que la tendencia actual es crear ordenadores con un mayor número de dispositivos electrónicos más pequeños con un gasto energético menor. El tamaño puede seguir disminuyendo, pero al toparnos con la cota inferior proporcionada por el principio de Landauer, el gasto energético no puede decrecer más.

La computación reversible nos ofrece una solución a este problema, aunque con algunas limitaciones. Una condición necesaria para la implementación de computación con coste que tiende a 0 es que el sistema en el que se realiza sea *adiabático*, es decir, que no intercambie calor con el exterior. La velocidad de computación genera desequilibrios que rompen esta condición, provocando efectos indeseables como la anteriormente comentada disipación de calor. Esta correlación es directa: cuanto menor sea la cota inferior de gasto energético, menor debe ser la velocidad de cómputo[FHA98][p. 152].

2.2.2. El problema del rendimiento

Desde los inicios de la computación, el avance en capacidad siempre ha estado definida por un aspecto clave: el número de operaciones o cálculos por unidad de energía disponible, o lo que es lo mismo, la eficiencia energética. Una forma de ver el rendimiento del proceso para completar una tarea es la siguiente:

$$R = \frac{N_{op}}{t} = \frac{N_{op}}{E_{dis}} \cdot \frac{E_{dis}}{t} = F_E \cdot P_{dis},$$

siendo R el rendimiento, N_{op} el número de operaciones necesarias para completar la tarea, t el tiempo necesario para completar dichas operaciones, E_{dis} el total de energía disipada en el proceso, $F_E = \frac{N_{op}}{E_{dis}}$ la eficiencia energética y $P_{dis} = \frac{E_{dis}}{t}$ la media de energía disipada a lo largo de la tarea[Fra05][p. 1].

A la hora de mejorar el rendimiento de un sistema, lo más importante y donde se invierte el mayor esfuerzo es, como ya hemos mencionado, en aumentar el número de operaciones realizadas por unidad energética que describimos con la variable F_E. En la actualidad se usan transistores tipo FET (Field Effect Transistor) en los que $F_E \approx \frac{1op}{\frac{1}{2}CV^2}$[Fra05][p. 1], siendo C la capacitancia de un nodo. La capacitancia es la relación entre la capacidad de acumulación energética de un condensador y el voltaje que existe en él. V es la variación máxima del voltaje que diferencia los dos estados posibles del transistor. Esta cantidad recibe el nombre de *voltaje lógico*. A esta cantidad se le asigna la variable E_s debido a que es la energía que acarrean las señales que codifican la lógica del circuito. Cada vez que se cambia el voltaje de un nodo o componente, se disipa una cantidad de energía E_s debido al mecanismo no reversible de la tecnología FET.

En las últimas décadas, la mejora de F_E se ha debido principalmente al decrecimiento de C, proporcional a la disminución en tamaño de los transistores actuales, debido a que el área de placas más pequeñas hace disminuir la capacitancia. Otro factor que ha contribuido a la mejora es la disminución de $5V$ a aproximadamente $1V$ del voltaje lógico.

El problema de esta situación es que parece estar llegándose a un callejón sin salida en lo que se refiere a la mejora del rendimiento. El tamaño de los transistores no puede disminuir sin que lo haga proporcionalmente el voltaje lógico debido a diversas formas de pérdidas y posible rotura de componentes. Por otra parte, el voltaje lógico también se encuentra limitado por las fugas producidas por la disipación de calor del propio componente. La energía térmica que produce un transistor es $E_T = k_B T$ siendo k_B la constante de Boltzmann and T la temperatura[Fra05][p. 2]. Esta energía proviene del fenómeno de borrado de información mencionado anteriormente y viene cuantificada por el principio de Landauer. Se ha

comprobado empíricamente que cuando la energía de señal E_s se acerca a la térmica, E_T , los dispositivos digitales dejan de poder cambiar de estado de forma consistente debido al ruido producido por la primera, provocando errores lógicos. Para que esto no ocurra, se ha comprobado que E_s debe permanecer alrededor de 100 veces mayor que E_T. Si $E_s \gtrapprox E_T$, la probabilidad de error se mantiene pequeña: alrededor de $e^{-100} = 3.72 \times 10^{-44}$[Frao5][p. 2].

En la actualidad las señales más débiles acarrean una energía $E_s = 100kT$. Siguiendo el patrón de mejora actual predicho por la ley de Moore, nos quedan unos 10 años de posibles mejores antes de alcanzar este límite físico permanente. Sin embargo, la computación reversible se nos presenta como alternativa para circunnavegar este problema[Frao5][p. 2].

Tomemos el límite inferior de la energía de señal $E_s = 100kT$[Frao5][p. 2]. Esta cantidad se refiere únicamente a la energía contenida en la señal lógica, es decir, es la cantidad necesaria para cambiar el estado del dispositivo electrónico. Sin embargo, nada nos obliga a disipar en forma de calor toda la energía contenida en la señal cuando esta es procesada de alguna manera, siempre que no se realice borrado de información. De hecho, es estas condiciones, no hay cota inferior conocida en cuanto a cantidad de energía que tenga que ser disipada en una operación dada.

En consecuencia, podríamos ser capaces, potencialmente, de recuperar parte de la energía contenida en la señal para futuras operaciones. De esta manera podríamos ser capaces de disipar una cantidad arbitrariamente pequeña de energía por operación, consiguiendo rodear el obstáculo que representa el principio de Landauer[Frao5][p. 2]. Por otra parte, también podríamos volver a incrementar la energía de la señal para aumentar su distancia respecto a la energía termal y así disminuir drásticamente la probabilidad de error producido por ruido térmico, sin temor al aumento proporcional de energía disipada.

2.3. Puertas lógicas

2.3.1. Definición y ejemplos simples

Es posible operar sobre bits de muchas formas distintas. La representación que utilizamos para expresar una operación sobre uno o varios bits se denomina *puerta lógica*. En realidad, las puertas lógicas son dispositivos electrónicos que nos permiten implementar lo que son, desde un punto de vista más formal, aplicaciones entre cuerpos que arrojan un valor lógico de verdad.

Definición 2.1. Se denomina *función booleana* a una aplicación f: $\mathbb{Z}_2^n \to \mathbb{Z}_2$ con $n \in \mathbb{N}$ y siendo $\mathbb{Z}_2$ el cuerpo formado por el conjunto 0,1 con la suma y el producto usual módulo 2.

Las funciones booleanas son los elementos fundamentales que componen las puertas lógicas.

Definición 2.2. Una *puerta lógica* es una aplicación $f\colon \mathbb{Z}_2^n \to \mathbb{Z}_2^m$ de manera que $f(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$ siendo f_i con $i = 1, \ldots, n$ funciones booleanas. Esta función a veces se le denomina *función booleana multivaluada*.

Ejemplo 2.2. La puerta lógica más básica es la denominada puerta NOT, que tiene como

función booleana:

$$f : \mathbb{Z}_2 \to \mathbb{Z}_2,$$
$$f(x) = \neg x = \overline{x} = x' = \begin{cases} 0 & x = 1, \\ 1 & x = 0. \end{cases}$$

Ejemplo 2.3. En otros casos podemos tener más de una variable de entrada o *input*. Una de las puertas lógicas más usuales es la puerta XOR, que tiene como función booleana:

$$\text{XOR} : \mathbb{Z}_2^2 \to \mathbb{Z}_2,$$
$$\text{XOR}(x, y) = x \oplus y = \begin{cases} 1 & x \neq y, \\ 0 & \text{en otro caso.} \end{cases}$$

Esta puerta se asocia a la suma en el cuerpo $\mathbb{Z}_2$.

Las puertas lógicas suelen llevar asociado una tabla donde se especifican todas las entradas posibles, con sus respectivas imágenes, llamada *tabla de verdad*. Esta tabla tiene 2^n entradas, siendo n el número de inputs.

Las puertas lógicas más utilizadas, junto con la NOT y la XOR, serán la AND y en menor medida la OR. La puerta lógica AND se corresponde al producto interno en $\mathbb{Z}_2$ y se simboliza lógicamente con $\wedge$, mientras que la puerta OR será algo distinta:

$$\text{OR} : \mathbb{Z}_2^2 \to \mathbb{Z}_2,$$
$$\text{OR}(x, y) = x + y = x \vee y = \begin{cases} 0 & x = y = 0, \\ 1 & \text{en otro caso.} \end{cases}$$

Definición 2.3. Una puerta lógica se dirá *reversible* si la aplicación que la define es biyecctiva. En otras palabras, si para cada entrada existe una única salida y viceversa.

A las funciones definidas de $\mathbb{Z}_2^n$ a $\mathbb{Z}_2^k$ con $n, k \in \mathbb{N}$ se denominan funciones (n, k). Las variables n y k hacen referencia al número de variables de salida y entrada, respectivamente. Para que una puerta lógica sea reversible, es necesario que el número de variables en el dominio o inputs sea el mismo que el número variables en el espacio de llegada o outputs, es decir $n = k$.

El hecho de que el número de variables de entrada y de salida tengan que ser las mismas representa una condición bastante restrictiva, ya que en la mayoría de los casos estos números distan de coincidir. Sin embargo, es posible modificar las puertas existentes para hacerlas reversibles añadiendo inputs o outputs.

Definición 2.4. Se denominan *auxiliares* a los outputs que se añaden a una puerta lógica con función (n, k) con $n \neq 0$ para transformarla en una con función (k, k). Por contrapartida se denominan *inputs constantes* a las variables de entrada que se le añaden a una puerta lógica para hacerla reversible. Otro nombre para estas variables de salida constantes es el de *valores ancillares cero* [Tah16][p. 7].

Ejemplo 2.4. A la puerta lógica XOR se le puede añadir una salida auxiliar para hacerla reversible de la siguiente manera:

$$f : \mathbb{Z}_2^2 \to \mathbb{Z}_2^2,$$
$$f(x,y) = (x, x \oplus y) = \begin{cases} (x,1) & x \neq y, \\ (x,0) & x = y \end{cases}$$

Que tiene por tabla de verdad:

in_1	in_2	out_1	out_2
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Lo cual claramente es una biyección y la puerta es, por lo tanto, reversible. A esta contraparte reversible de la puerta XOR se le denomina *puerta de Feynman* o CNOT (controlled not), ya que dependiendo del estado del primer bit, x, se le aplica una negación al segundo. Por otra parte, también es posible que tengamos que añadir tanto una entrada como una salida adicional.

Ejemplo 2.5. Si intentamos construir la contraparte reversible de la puerta lógica AND, obtenemos otra extremadamente importante, llamada puerta *Toffoli*. Cuya función se define por:

$$T : \mathbb{Z}_2^3 \to \mathbb{Z}_2^3,$$
$$T(x_1, x_2, x_3) = (x_1, x_2, (x_1 x_2) \oplus x_3)$$

Esta puerta también es llamada CCNOT (controlled-controlled-NOT) puesto que el hecho de cambiar el tercer bit depende del producto de los dos primeros. En general, las puertas se pintan dentro de circuitos como cajas con cables entrando y saliendo de ellas, pero en el caso de la puerta Toffoli, se representan de manera diferente.

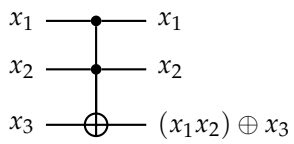


Figura 2.1: Puerta Toffoli con 3 entradas.

Ejemplo 2.6. Veamos ahora la puerta SEMISUMADOR. Esta puerta consta de dos variables de entrada y dos de salida, computando el equivalente a la suma con resto: en la primera salida calcula $in_1 \oplus in_2$ y en la segunda $in_1 \cdot in_2$. Estas operaciones arrojan la tabla de verdad:

in_1	in_2	out_1	out_2
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	1

Como podemos ver, esta función no es biyectiva, ya que 10 y 01 comparten imagen. Esto puede remediarse añadiendo lo que se denomina *almacenamiento temporal*, de modo que definimos:

$$f : \mathbb{Z}_2^3 \to \mathbb{Z}_2^3,$$
$$f(x_2, x_1, x_0) := (y_2, y_1, y_0) = (x_2, x_2 \oplus x_1, x_0 \oplus (x_1 \cdot x_2)).$$

Esta nueva puerta se denomina *Peres* y su función es claramente biyectiva, ya que tenemos x_1 gracias al valor de x_2 y $x_1 \oplus x_2$, siendo esta última operación reversible. Del mismo modo podemos conseguir x_0. Si siempre entra por x_0 una constante 0, esta puerta tiene una función equivalente a la de SEMISUMADOR.

En esta nueva puerta se hace uso de un par input-output, (x_2, y_2), sobre el que no se aplican operaciones. Su valor solo se lee y nunca se sobrescribe. A esta pareja de valores que transporta información que no se altera se le denomina *almacenamiento temporal*[Tah16][p. 8].

2.3.2. Universalidad

En general, existen una variedad de maneras para hacer reversible una puerta lógica, siendo el más simple de estos métodos el llevar una copia de todos los inputs a la salida mediante almacenamiento temporal. Sin embargo, este método es poco práctico y bastante costoso en cuanto a utilización de componentes.

Las puertas lógicas, en general, por sí mismas, no son demasiado útiles, pero sí que lo son cuando se utilizan iterativamente para hacer cálculos más complejos. Esta composición de operaciones se denomina circuito y puede representarse gráficamente como puertas lógicas unidas por cables que transmiten la información de unas a otras. Por ejemplo:

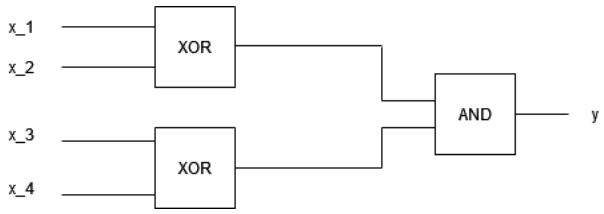


Figura 2.2: Ejemplo de circuito simple

Se trata de un circuito básico que calcula la función:

$$f : \mathbb{Z}_2^4 \to \mathbb{Z}_2$$
$$f(x_1, x_2, x_3, x_4) = (x_1 \oplus x_2) \cdot (x_3 \oplus x_4).$$

Esta función claramente no es biyectiva, por lo que este circuito es no reversible. ¿Cuál sería el equivalente reversible de este circuito? En primer lugar necesitaríamos cambiar las puertas XOR por las puertas de Feynman, y la AND por una Toffoli, que son sus contra partes reversibles, quedando:

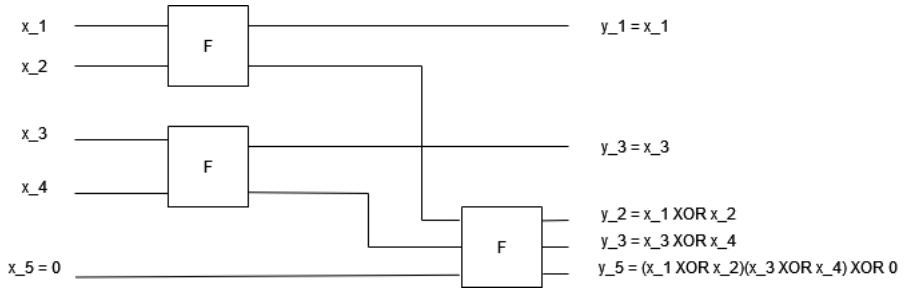


Figura 2.3: Equivalente reversible de 2.2

Claramente, el conjunto de inputs y outputs ha crecido. Este circuito es equivalente porque cuando consideramos las variables $x_1, x_2, x_3, x_4, x_5 = 0$ e y_5, la tabla de verdad que obtenemos es la misma. Hay muchísimos circuitos reversibles que, bajo las mismas condiciones, pueden ser equivalentes al original no reversible. Sin embargo, siempre será obviamente más deseable aquel que sea el más simple de estos, tanto como por consumo energético, como por escalabilidad y corrección de errores.

Al ver la transformación que ha sufrido este circuito tan simple, cabe preguntarse cómo es posible el diseño de circuitos reversibles cuando las operaciones son infinitamente más complejas que en este caso básico. De hecho, ¿cómo sabemos si siquiera es posible realizar siempre este proceso? En otras palabras, ¿es posible encontrar siempre un circuito reversible que realice los mismos cálculos que un circuito no reversible dado? La respuesta a esta pregunta viene dada por la propiedad de *completitud funcional*. Para poder llegar a este concepto necesitaremos algunas definiciones previas.

Definición 2.5. La *aridad* de una función booleana es la dimensión del espacio de salida.

Ejemplo 2.7. La aridad de la función AND es 2, puesto que se trata de una función que va de $\mathbb{Z}_2^2$ a $\mathbb{Z}_2$.

Definición 2.6. Sea $\mathbb{Z}_2^n, n \in \mathbb{N}$ cualquiera. Un conjunto de funciones booleanas C se dice que es un *clon* en $\mathbb{Z}_2^n$ si se cumplen las siguientes condiciones:

- C contiene todas las proyecciones posibles, es decir, $\forall k, n \in \mathbb{N}$, la función:

$$\pi_k^n : \mathbb{Z}_2^n \to \mathbb{Z}_2,$$
$$\pi_k^n(x_1, \ldots, x_n) = x_k$$

  está en C.

- La composición de cualquier conjunto de funciones booleanas F= $\{p, f_1, \ldots, f_m\}$ tal

que p tiene una aridad m y f_i con $i = 1, \ldots, m$ tiene una aridad n,entonces la función:

$$h : \mathbb{Z}_2^n \to \mathbb{Z}_2,$$
$$h(x_1, \ldots, x_n) = p(f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$$

está en C.

Esta definición nos permite formalizar el concepto de construcción de una función a partir de otras en el contexto de las puertas lógicas. Dado un clon C, se dice que un conjunto de funciones booleanas F es capaz de generarlo cuando cualquier función de C puede escribirse como composición de funciones de F y cualquier cantidad de proyecciones π_k^n.

Definición 2.7. Un conjunto F de funciones boolenas se dice que es *funcionalmente completo* si cualquier función booleana $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ con $n \geq 1$ pertenece al clon generado por F.

Proposición 2.1. *El clon, C, generado por* AND, NOT *es funcionalmente completo.*

Demostración. Puede verse a partir de una demostración sencilla. En primer lugar, vemos que la función:

$$f : \mathbb{Z}_2^2 \to \mathbb{Z}_2,$$
$$f(x_1, x_2) = \neg((\neg x_1) \wedge (\neg x_2))$$

arroja la tabla de verdad:

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

Esta tabla es precisamente la correspondiente a la función OR, así que OR $\in C$. Consideremos ahora una función booleana f cualquiera con aridad n y una tabla de verdad:

$x_1 \ldots x_2$	y
$a_{11} \ldots a_{1n}$	b_1
$a_{21} \ldots a_{2n}$	b_2
$\ldots$	$\ldots$
$a_{2^n 1} \ldots a_{2^n n}$	b_{2^n}

Ahora, para cada fila de esta tabla que cumpla $b_i = 1$ podemos construir una función equivalente a f solo en esa fila. Para ello, llamamos I al conjunto de índices de las filas tal que $b_i = 1$, le aplicamos una negación a cada $a_i = 0$ y después aplicamos AND a todos los resultados.

Por ejemplo, si f cumple en la fila 3 que $f(1, 0, 0, 1, 0) = 1$, entonces construimos:

$$f_5 = a_1 \wedge \neg a_2 \wedge \neg a_3 \wedge a_4 \wedge \neg x_5.$$

A continuación solo queda construir $f' = \vee_{i \in I} f_i := f_{i_1} \vee \cdots \vee f_{i_n}$. De esta manera, hemos construido una función f' que da como resultado 1 únicamente cuando las variables de entrada se asemejen a alguna de las filas en las que f sea 1, consiguiendo así una función con la misma tabla de verdad y por lo tanto, equivalente. Con esto concluimos que $f \in$C.

□

Cabe destacar que la forma en la que hemos construido la función booleana utilizada en la demostración se denomina *forma normal conjuntiva*. Esta forma siempre puede construirse a partir de una tabla de verdad. Una función booleana se dice que está en forma FNC cuando se expresa en forma de productos de variables y sus negaciones a los que se aplica la puerta OR. Esta operación se representa mediante $\vee$ o +. El primer símbolo se utiliza en contextos de lógica más formal, mientras que el segundo hace referencia a la puerta.

Ejemplo 2.8. Sea una función booleana $f : \mathbb{Z}_2^2 \to \mathbb{Z}$ que tiene como tabla de verdad:

x_1	x_2	y
0	0	1
0	1	1
1	0	0
1	1	0

Y puede expresarse en FNC como:

$$f(x_1, x_2) = \overline{x_1 x_2} + \overline{x_1} x_2.$$

Ahora sabemos que es posible construir cualquier puerta únicamente mediante puertas AND y NOT. Sin embargo, estas puertas no son reversibles y cabe preguntarse si existe la posibilidad de que un conjunto de puertas lógicas reversibles pueda generar un conjunto funcionalmente completo. La respuesta es que sí.

Proposición 2.2. *El clon C generado por la puerta Toffoli es funcionalmente completo.*

Demostración. La puerta Toffoli tiene como función asociada:

$$T : \mathbb{Z}_2^3 \to \mathbb{Z}_2^3,$$
$$T(x_1, x_2, x_3) = (y_1, y_2, y_3) = (x_1, x_2, (x_1 x_2) \oplus x_3)$$

Que tiene como tabla de verdad:

x_1	x_2	x_3	y_1	y_2	y_3
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	1	0

Podemos construir la puerta AND fijando $x_3 = 0$ y proyectando en x_3, mientras que la puerta NOT puedo construirse fijando $x_1 = x_2 = 1$ y de nuevo, proyectando en x_3. Es decir:

$$\pi_3^3 \circ T(x_1, x_2, 0) = \pi_3^3(x_1, x_2, x_1 x_2) = x_1 x_2 = \text{AND}(x_1, x_2)$$
$$\pi_3^3 \circ T(1, 1, x) = \pi_3^3(1, 1, x_1 \oplus 1) = x \oplus 1 \equiv \neg x = \text{NOT}(x)$$

Como AND y NOT son generadores de un conjunto funcionalmente completo y hemos demostrado que que AND y NOT están en el clon generado por la puerta Toffoli, entonces claramente dicho clon es funcionalmente completo. □

Este resultado nos ofrece una conclusión de lo más valiosa: para cada función booleana existe un circuito reversible que lo implementa. Sin embargo, a priori no sabemos cuál puede ser. Construir dicho circuito ha sido materia de investigación durante décadas, obteniendo una variedad de métodos de síntesis, cada uno con sus propias características.

2.4. Métodos para la síntesis de circuitos reversibles

Como ya hemos podido comprobar, la lógica reversible es profundamente contra intuitiva y la restricción que presenta el hecho de tener que conservar toda la información tras cualquier operación representa un gran obstáculo en cuanto a diseño de circuitos que sean escalables. Las heurísticas o directrices que se consideran deseables para el diseño son:

- No cree muchas salidas de compuertas y subcircuitos.
- Estas salidas se reutilizan como entradas de otras puertas.
- El número total de constantes en las entradas de las puertas se mantiene lo más bajo posible.
- Si se necesitan dos copias de una señal, se utiliza un circuito de copia (puerta Feynman).
- El circuito resultante es acíclico, lo que significa que no puede haber bucles.
- El método debe ser de aplicación general[Tah16][p. 7].

En primer lugar presentaremos algunas definiciones necesarias.

Definición 2.8. Una función $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ se denomina *trasposición* si intercambia dos elementos cualquiera de posición, es decir:

$$f(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) = (x_1, \ldots, x_j, \ldots, x_i, \ldots, x_n)$$

para algún i,j ∈1,. . . ,n.

Definición 2.9. Una función $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ se denomina *permutación* si puede escribirse como composición finita de trasposiciones.

Definición 2.10. Una permutación se dice *par* si existe una composición de un número par de trasposiciones que la implementan.

Definición 2.11 (Generalización de la puerta Toffoli)**.** Es posible generalizar la puerta Toffoli a dimensión n cualquiera, obteniendo:

$$\text{TOF}_n^n(x_1, \ldots, x_n) = (x_1, \ldots, x_{n-1}, (\prod_{i=0}^{n-1} x_i) \oplus x_n)).$$

Por otra parte, la variable de salida sobre la que se aplica la operación XOR no tiene por qué ser la última. Esta variable puede ser cualquier x_i con $1 \leq i \leq n$, obteniendo así la función:

$$\text{TOF}_i^n(x_1, \ldots, x_n) = (x_1, \ldots, (\prod_{j \neq i} x_i) \oplus x_j), \ldots, x_n).$$

El caso $n = 1$ nos da la puerta NOT. Por otro lado, al ser el número de variables cambiante, necesitamos dos nuevas definiciones: el índice que recibe la operación XOR se denomina *salida objetivo*, mientras que aquellos que aparezcan multiplicados en el otro sumando de la suma módulo 2 son llamados *entradas de control*.

2.4.1. Método de síntesis exacta

Este método es uno de los más simples conceptualmente, pero tiene una restricción: solo puede utilizarse para crear circuitos que implementen permutaciones pares.

En primer lugar, planteamos el problema de decisión, que está parametrizado por f y d como: ¿es posible implementar la función booleana f con un circuito conformado por d puertas Toffoli?.

Toda función booleana multivaluada reversible puede construirse con puertas Toffoli, como ya hemos demostrado anteriormente, así que este problema siempre devuelve sí para algún d. Por tanto, se trata de un problema *decidible*.

En primer lugar, se crea una codificación del problema, para así transformarlo en un problema de *satisfibilidad booleana* (SAT).

Definición 2.12. Se denomina SAT al problema de decisión:
Dada una expresión booleana ¿existe una asignación para las variables que componen dicha expresión de manera que devuelva un resultado afirmativo?

La idea principal será codificar el problema de decisión original para transformarlo en un SAT. Para ellos construimos una serie de vectores. Dada una función reversible f con n variables de entrada, tenemos:

- $t^k = (t_j^k, \ldots, t_1^k)$. Con $0 \leq k \leq d$, $0 \leq j \leq E(log_2(n)) + 1$ (a menos que el logaritmo sea exacto, en cuyo caso no sumamos 1), siendo E la función parte entera. Este vector representa una codificación binaria de un entero que simboliza el índice de la variable de salida objetivo de la puerta Toffoli a una profundidad k. Se entiende profundidad k como la k-ésima puerta.
- $c^k = (c_{n-1}^k, \ldots, c_1^k)$ con $0 \leq k \leq d$. Este vector está formado por variables binarias representando las variables de control de la siguiente forma: si $c_i^k = 1$ entonces el índice $(t^k + i) \bmod n$ es una línea de control de la puerta lógica de profundidad k, cuya línea objetivo es t^k.

- $x_i^k = (x_{i_n}^k, \ldots, x_{i_1}^k)$. Con $0 \leq i \leq 2^n - 1$ y $0 \leq k \leq d$. Este vector representa el estado de las variables de salida de la k-ésima o las variables de entrada de la $k+1$-ésima puerta lógica, para la combinación inicial de la fila i de la tabla de verdad.

Dada esta codificación del problema, transformamos el problema original en uno equivalente, que llamamos $SAT(f, d)$: ¿existe una asignación para los vectores t^k e c^k para cada $k = 1, \ldots, d$, de manera que x_i^0 es igual a la fila i de las variables de entrada e x_i^d es el resultado i de la tabla de verdad de f?

Ahora, se aplica un enfoque de "fuerza bruta". Empezando desde $d = 1$, probamos asignaciones de los vectores t y c hasta conseguir una asignación que satisfaga el problema $SAT(f, d)$ para una f y d determinadas. Si no existe tal asignación, incrementamos d y reiniciamos la búsqueda.

Obviamente, este enfoque nos proporciona siempre el circuito con el menor número posible de puertas. Sin embargo, es bastante lento. Es por este motivo que existen métodos de aproximación de cotas que ofrecen buenos resultados. Podría pensarse que, dada una primera respuesta afirmativa del problema para d determinado, es suficiente comprobar que no existe asignación para $d - 1$ para concluir que no es posible construir una implementación con $d - 1$ puertas, pero esto es erróneo.

Teorema 2.2. *Sea $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ una función reversible que desea implementarse. La implementación con el menor número de puertas lógicas es d si no existen implementaciones con $d - 1$ y $d - 2$ puertas lógicas.*

Efectivamente, este teorema nos dice que es necesario comprobar las iteraciones con dos unidades menos del algoritmo[AKS20][p. 70].

Un claro problema de este algoritmo es que el espacio de búsqueda de las asignaciones, o dicho en otras palabras, la cantidad potencial de asignaciones posibles aumenta exponencialmente cuando incrementamos d, luego la ejecución es bastante costosa. Es por este motivo que generalmente este algoritmo solo se continúa hasta $d \approx 10$, antes de adoptar otras vías[AKS20][p. 75].

2.4.2. Método basado en transformación

Consideramos una función binaria multivaluada reversible cualquiera con aridad N. Este método usa la reversibilidad de la puerta Toffoli en su beneficio para construir, de forma iterativa, un circuito que tome las salidas de una función dada y lo que coloquialmente podría considerarse ïnverso", ya que a partir de cualquier $f(i)$, intentamos hacer transformaciones con puertas Toffoli hasta llegar a i, siendo i las representaciones binarias de los enteros desde 0 hasta $2^N - 1$.

En este algoritmo, construiremos un circuito paso a paso, que guardaremos en una variable llamada C, generando como máximo $n2^{n-1}$ puertas lógicas[MMD03][p. 3]. El desarrollo del algoritmo es el siguiente:

1. Si $f(0) = 0$, entonces no se hace nada. Si $f(0) \neq 0$, entonces realizamos una transformación a la salida de $f(0)$ hasta transformarla en 0 y esta pasa a ser la nueva función

con la que trabajamos. Esta inversión requiere un número de puertas TOF_1^n igual al número de bits que necesiten ser cambiados.

2. Para cada $1 \leq i \leq 2^m - 1$, tomamos $f(i)$ y si cumple que $f(i) = i$ no hacemos nada, pero si $f(i) \neq i$, entonces negamos los bits de la salida necesarios para que así sea. En primer lugar, definimos antes de cada transformación:
   - p = numero binario de n cifras que tiene un 1 en aquellas posiciones j tales que $i_j = 1$ pero $f(i)_j = 0$.
   - q = numero binario de n cifras que tiene un 1 en aquellas posiciones j en las que $i_j = 0$ pero $f(i)_j = 1$.

   Para cada i:
   - Calculamos p y q.
   - Empezando por p, si nos encontramos una posición j en la que haya un 1, añadimos a C una puerta Toffoli que tiene como entradas de control aquellas posiciones de $f(i)$ que sean 1 y su salidas objetivo es j. Después, recalculamos p y q.
   - si $p = 0 \ldots 0$, entonces consideramos q. Si nos encontramos un 1, añadimos a C una puerta Toffoli que tiene como entradas de control aquellas posiciones en las que $f(i)$ sea 1 y como salida objetivo el índice de p que estemos considerando.

En esencia, lo que hacemos es construir un circuito de forma iterativa concatenando puertas Toffoli que nieguen los bits que obstaculizan el conseguir el objetivo propuesto: $f_{transfor}(output_i) = i$.

Ejemplo 2.9. Para esclarecer un poco el proceso del algoritmo, presentaremos una ejecución del algoritmo para la función reversible cuya tabla de verdad es la siguiente:

$x_1x_2x_3$	$y_1y_2y_3$
000	001
001	000
010	011
011	010
100	101
101	111
110	100
111	110

En primer lugar, tomamos $i = 0$. Como $f(0) = 1$, aplicamos una puerta NOT, os lo que es lo mismo, añadimos a C la puerta $\text{TOF}_1^3(x_3)$. Tras componer f con esta puerta, la tabla de verdad pasa a ser:

$x_1x_2x_3$	$y_1^1y_2^1y_3^1$	$y_1^2y_2^2y_3^2$	$y_1^3y_2^3y_3^3$	$y_1^4y_2^4y_3^4$
000	00**0**	000	000	000
001	00**1**	001	001	001
010	01**0**	010	010	010
011	01**1**	011	011	011
100	10**0**	100	100	100
101	11**0**	10**1**	**10**1	101
110	10**1**	101	**11**1	110
111	11**1**	11**0**	110	111

Convenientemente, $f(i) = i$ para $0 \leq i \leq 4$, luego no hacemos nada para esas varibles. A continuación, para $i = 5$, calculamos p y q, obteniendo:

- $p = 001$
- $q = 010$

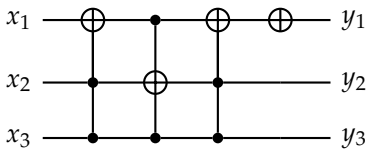
Para $j = 3$ tenemos que $p_j = 1$ luego añadimos a C una puerta Toffoli T_3^3 que tiene como entradas de control x_1 y x_2 y salida objetivo x_3, quedando la columna de superíndice 1. Ahora recalculamos p y q, que dan:

- $p = 010$
- $q = 001$

Como $i_j = 1$. Aplicamos una puerta Toffoli T_2^3 que tiene como entradas de control x_1 y x_3 y salida objetivo x_2. Tras apriclar dicha puerta y añadirla a C, nos queda la tabla con superíndice 3. En el siguiente paso, para $i = 6$, calculamos p y q:

- $p = 000$
- $q = 001$

Como $q_3 = 0$, aplicamos TOF_3^3 similar a la de la columna 1, obteniendo así la columna 4. Tras aplicar esta puerta se cumple que , $f(i) = i$ para $i = 1, \ldots, 7$, terminando así el algoritmo. El circuito está guardado en C. Para obtener el resultado, bastará con aplicar las puertas en orden inverso, obteniendo el circuito:



Esta imagen presenta una forma equivalente de escribir la puerta Toffoli, siendo la línea con el símbolo $\oplus$ la salida objetivo y las otras las entradas de control.

2.5. Antesala a la computación cuántica

La reversibilidad en la computación, como propiedad fundamental que permite deshacer las operaciones computacionales y recuperar la información original sin pérdida de energía, ha emergido como un paso previo crucial en el camino hacia la implementación práctica de la computación cuántica.

De hecho, teniendo en cuenta los elementos más básicos de la computación cuántica, es decir, el qubit, tenemos que considerar que este es solo una representación abstracta de un sistema físico de la misma manera que el bit representa el paso de corriente o no. Es por esto que todas las operaciones deben ser reversibles, puesto que los procesos cuánticos que aprovechamos para realizar esta nueva etapa en la computación lo son: en el mundo real no

se pierde información. Es por esto que todas las puertas lógicas utilizadas en la computación cuantica son reversibles.

La reversibilidad en la computación cuántica es especialmente relevante en la implementación de algoritmos cuánticos, que son la base de la capacidad de procesamiento superior de los sistemas cuánticos en comparación con los sistemas clásicos. Los algoritmos cuánticos, como el algoritmo de Shor para la factorización de números enteros y el algoritmo de Grover para la búsqueda en bases de datos no estructuradas, se basan en la manipulación precisa de estados cuánticos y la explotación de fenómenos cuánticos, como la superposición y el entrelazamiento. La reversibilidad de las operaciones cuánticas es esencial para mantener la integridad de los estados cuánticos durante la ejecución de estos algoritmos, lo que garantiza la obtención de resultados precisos y confiables.

En definitiva, la reversibilidad en la computación cuántica es un aspecto fundamental que juega un papel crucial en la garantía de la integridad y confiabilidad de los resultados computacionales en sistemas cuánticos. Permite la manipulación precisa de estados cuánticos, la implementación de algoritmos cuánticos, la corrección de errores y la optimización de arquitecturas de hardware cuántico. La reversibilidad es un paso previo esencial hacia la computación cuántica, ya que contribuye a la superación de los desafíos inherentes a la decoherencia y la sensibilidad cuántica de los sistemas, y abre el camino para el desarrollo de sistemas cuánticos más confiables y eficientes. A medida que avanzamos en el campo de la computación cuántica, la reversibilidad seguirá siendo un área de investigación y desarrollo clave para lograr avances significativos en la capacidad de procesamiento y aplicaciones prácticas de la computación cuántica en diversos campos, como la criptografía y la simulación de sistemas cuánticos complejos.

3 Fundamentos de la mecánica cuántica

La mayor parte de los conceptos que se discuten en este capítulo provienen de la fuente [NC10].

3.1. Introducción

La mecánica cuántica es una teoría revolucionaria que se originó a principios del siglo XX para explicar los fenómenos observados en el mundo subatómico. A diferencia de la física clásica, que se ocupa de objetos macroscópicos, la mecánica cuántica se aplica a partículas extremadamente pequeñas, como electrones, protones y fotones. La teoría cuántica tiene implicaciones profundas, como la idea de que una partícula puede estar en múltiples lugares al mismo tiempo, o que medir una propiedad de una partícula puede cambiar instantáneamente el estado de otra partícula en otro lugar. A pesar de su complejidad y de que desafía la intuición humana, la mecánica cuántica es una herramienta poderosa para describir procesos que a priori parecen totalmente contra intuitivos.

Una de las características principales de la mecánica cuántica es el no determinismo. El estado de un sistema físico cuántico obedece al principio de indeterminación de Heisenberg, que cuantifica la imposibilidad de medir con total precisión magnitudes como la posición y la velocidad. En lugar de esto, la mecánica cuántica describe la naturaleza probabilística de los eventos en el mundo subatómico, lo que significa que no se puede predecir con certeza el resultado de un experimento en particular, sino solo la probabilidad de cada resultado posible.

En este capítulo describiremos los fenómenos más importantes de los que saca provecho la computación cuántica para conseguir mejoras en eficiencia. Uno de ellos es la superposición, un fenómeno en el cual una partícula puede estar en varios estados a la vez. Otro es el entrelazamiento cuántico: cuando dos partículas quedan entrelazadas, manteniendo una dependencia en sus estados sin importar el tiempo que pase o la distancia entre estas.

El sistema más fundamental que nos atañe será aquel que se compone de una sola partícula. Este sistema tiene asociado un espacio vectorial complejo de dimensión 2 que lo describe totalmente. Si tomamos una base ortonormal $B = \{v_0, v_1\}$, podemos realizar la identificación $v_i = |i\rangle$, de modo que un vector unitario cualquiera se expresa como:

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle\,, \tag{3.1}$$

siendo a, b escalares complejos que cumplen $|a|^2 + |b|^2 = 1$. Llamamos al vector $|\psi\rangle$ *estado* del sistema y a, b las amplitudes de los vectores $|0\rangle$ y $|1\rangle$ respectivamente.

Este sistema se denomina *qubit*, que es un acortamiento de *quantum bit*, y será de vital importancia. De la misma manera que el bit es la unidad más básica en el marco de tratamiento de información convencional, el qubit será su equivalente cuántico: las unidades fundamentales que se unirán para formar sistemas más complejos. La diferencia entre estas dos unidades es que el bit es una unidad de información que puede ser o bien 0 o bien 1 mientras que el qubit, como vemos en la ecuación (3.1), puede estar en cualquier combinación lineal de los estados que hayamos elegido como base. Esta expresión es la representación matemática de la *superposición*. Comentaremos este fenómeno con más profundidad más adelante.

Por otro lado, sabemos que es posible reescribir la ecuación 3.1 como:

$$|\psi\rangle = e^{i\gamma}(\cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle),$$

siendo γ ,φ y θ números reales.

En esta ecuación, $e^{i\gamma}$ se denomina *factor de fase global*. La fase global debe su nombre al hecho de que se aplica a todas las amplitudes de los vectores de la base elegida que componen el estado. Por ejemplo, los estados $|\psi\rangle$ y $e^{i\theta}|\psi\rangle$ se dice que se diferencian en un factor de fase global. Esta fase es irrelevante desde un punto de vista estadístico y no afecta a la medición.

Por otro lado, al escalar $e^{i\varphi}$ se le denomina *factor de fase relativa* puesto que solo afecta a una de las amplitudes de los vectores de la base que componen el estado. Se dice que dos estados difieren en un factor de base relativo si $\exists\theta \in \mathbb{R}$ tal que $a = e^{i\theta}a'$, siendo a y a' amplitudes del mismo vector de la base en los distintos estados. Como es un escalar que multiplica a una de las coordenadas del vector de estado, es dependiente de la base. El factor de fase relativo , al contrario que el global, si que es relevante en la medición y no puede ignorarse.

Ejemplo 3.1. Los estados:

$$a|0\rangle + b|1\rangle, \qquad a|0\rangle - b|1\rangle,$$

difieren en una fase relativa de -1 puesto que la amplitud de $|1\rangle$ en el segundo estado es $e^{i\pi}$ veces la amplitud de $|1\rangle$ en el primero.

Volvamos a la ecuación 3.1. Ahora que sabemos que el factor de fase global no es significativo, podemos reescribir la ecuación sin él:

$$\cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle.$$

Los números reales φ y θ representan las coordenadas esféricas de un punto en la esfera real, por lo que es posible representar un estado del qubit como un punto en dicha esfera. A esta esfera se la denomina *esfera de Bloch* y es un concepto importante puesto que permite visualizar el estado cuántico de un qubit de manera fácil y eficiente en un espacio tridimensional.

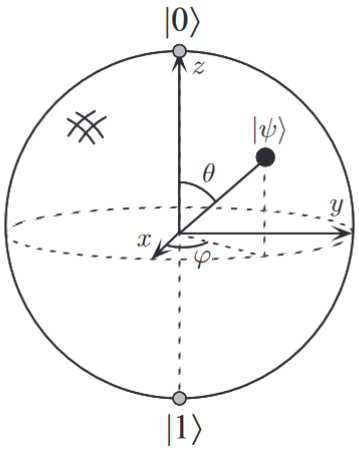


Figura 3.1: Esfera de Bloch. Como hemos indicado φ y θ representan los ángulos que codifican un punto en la esfera.

Existen infinitos estados posibles para un qubit (tantos como puntos en la esfera), pero como veremos en el postulado 3.4, los resultados de una medición determinada son finitos. A continuación presentaremos los 4 postulados fundamentales que definen el comportamiento de los sistemas cuánticos.

3.2. Postulados de la mecánica cuántica

La mecánica cuántica tiene como base 4 postulados (concepto físico equivalente al axioma matemático) que sientan las "reglas de juego" que seguiremos cuando exploremos los conceptos de este capítulo.

Postulado 3.1 (Espacio de estados)**.** *Cada sistema físico aislado tiene asociado un espacio de Hilbert denominado espacio de estados. En un instante de tiempo dado, el sistema queda totalmente determinado por el vector de estado, que es un vector unitario en el espacio de estados.*

Una vez que hemos definido el estado de un sistema físico dado, cabe preguntarse como evoluciona con el tiempo.

Postulado 3.2 (Evolución en tiempo discreto)**.** *La evolución de un sistema físico cerrado está descrita por una operación unitaria. Dicho de otra manera, si $|\phi\rangle$ es el vector de estado en el instante t_1, entonces el vector de estado en el instante t_2 es $U|\phi\rangle$, siendo U una matriz unitaria que solo depende de t_1 y t_2.*

De nuevo, la mecánica cuántica no nos dice cual es la matriz U, solo nos dice que la transformación del sistema sigue este patrón. Este postulado describe cual es la diferencia entre dos vectores de estado en instantes concretos pero ¿Que ocurre cuando consideramos el tiempo como una variable continua? Existe una versión alternativa del postulado anterior para este caso.

Postulado 3.3 (Evolución en tiempo continuo)**.** *La evolución de un sistema físico cerrado está descrita por la ecuación diferencial de Schrödinger:*

$$i\hbar \frac{d\,|\psi\rangle}{dt} = H\,|\psi\rangle\,.$$

En esta ecuación $\hbar$ es conocida como la constante de Plank, que debe calcularse experimentalmente. H es una matriz Hermitiana fija conocida como el Hamiltoniano del sistema.

En general, este último postulado no producirá ninguna utilidad operativa y nos centraremos en el 3.2. Este postulado es el motivo por el cual aplicar puertas lógicas cuánticas a un sistema físico tiene sentido: estas puertas representan una operación unitaria que se corresponde a una transformación válida del sistema. Es obviamente necesario que la representación matemática que utilicemos para describir un sistema físico guarde correspondencia con los procesos que ocurren en el sistema real. Este postulado nos garantiza que al aplicar puertas lógicas como la Hadamard o la Toffoli, esta correspondencia entre modelo y realidad se conserva.

Ya hemos descrito el estado y la transformación de un sistema físico. A continuación pondremos nuestra atención en la *medición*. ¿Cómo es posible que al medir una partícula, el resultado de la medición produzca un valor esperable y no un valor aleatorio? La respuesta a esta pregunta viene dada por el próximo postulado.

Postulado 3.4 (Medición cuántica)**.** *Cualquier medición de un sistema físico cuántico queda descrito por una colección finita de operadores $\{M_m\}$ denominados operadores de medida. Estos operadores actúan sobre el espacio de estados. Si el estado del sistema cuántico es $|\psi\rangle$, entonces la probabilidad de obtener el resultado m tras la medición es:*

$$p(m) = \langle\psi|\, M_m^\dagger M_m\,|\psi\rangle\,.$$

En caso de ser m_0 el resultado de la medición, entonces el estado del sistema, $|\psi'\rangle$, tras realizar dicha medición es:

$$|\psi'\rangle = \frac{M_{m_0}\,|\psi\rangle}{\sqrt{\langle\psi|\, M_{m_0}^\dagger M_{m_0}\,|\psi\rangle}}. \tag{3.2}$$

Además, el conjunto $\{M_m\}$ cumple la denominada ecuación de completitud*:*

$$\sum_i M_m^\dagger M_m = I \implies \sum_i \langle\psi|\, M_m^\dagger M_m\,|\psi\rangle = 1. \tag{3.3}$$

Este postulado es uno de los conceptos más fundamentales y esenciales en la mecánica cuántica. Establece la forma en que se pueden obtener observaciones cuantitativas a partir de sistemas cuánticos. En esencia, este postulado afirma que la medida de una propiedad cuántica en un sistema en un estado cuántico dado dará como resultado uno de los valores posibles de esa propiedad con ciertas probabilidades. Esto es debido a que la ecuación (3.3) expresa que el conjunto de los subíndices m son todos los resultados posibles. Si no fuese así, la suma de las probabilidades sería menor que 1. Sin este postulado, no sería posible predecir los resultados de las mediciones cuánticas y, por lo tanto, no sería posible hacer predicciones sobre el comportamiento de los sistemas cuánticos.

Por otra parte, este postulado pone de manifiesto una propiedad interesante y enigmática: la medición de un sistema afecta al mismo. Si $|\psi\rangle$ es el estado del sistema físico, entonces la ecuación (3.2) nos dice cómo quedará tras realizar una medición determinada. Este hecho nos permite inferir que aunque un sistema pueda estar en muchos estados a la vez y codificar una cantidad de información muy grande debido al fenómeno de superposición, solo podremos acceder a parte de ella.

Debido a que el postulado sobre medición cuántica se explica de manera bastante abstracta, procederemos con un ejemplo que ponga de manifiesto la utilidad operativa del mismo.

Ejemplo 3.2. Supongamos un sistema cuántico formado por una sola partícula. Ya hemos visto que este sistema queda descrito por un espacio vectorial complejo de dimensión dos. Fijemos la base $B = \{|0\rangle, |1\rangle\}$ con $|0\rangle = [1,0]^T$ y $|1\rangle = [0,1]^T$. Definamos ahora el conjunto de operadores de medición:

$$M_0 = |0\rangle\langle 0| = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \qquad M_1 = |1\rangle\langle 1| = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

Como vimos en el capítulo 1, las matrices M_0 y M_1 son proyectores sobre los subespacios vectoriales generados por sus respectivos vectores. Estos operadores cumplen que $M_m^\dagger = M_m$ y además $M_m^2 = M_m$ con $m = 1, 2$. Por otra parte, también complen la ecuación de completitud, ya que $M_0^\dagger M_0 + M_1^\dagger M_1 = I$. Por lo que, si el estado actual del sistema es $|\psi\rangle = a|0\rangle + b|1\rangle$, entonces se da que las probabilidades de obtener $|0\rangle$ y $|1\rangle$ tras medir son:

$$p(0) = (a\langle 0| + b\langle 1|)|0\rangle\langle 0|(a|0\rangle + b|1\rangle) = a^2,$$
$$p(1) = (a\langle 0| + b\langle 1|)|1\rangle\langle 1|(a|0\rangle + b|1\rangle) = b^2.$$

Además, podemos ver que si hemos obtenido el valor 0 en la medición, el estado del sistema, $|\psi'\rangle$ queda:

$$|\psi'\rangle = \frac{M_0|\psi\rangle}{\sqrt{\langle\psi|M_0^\dagger M_0|\psi\rangle}} = \frac{\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}}{\sqrt{\begin{bmatrix} a & b \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}}} = \frac{a}{|a|}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{a}{|a|}|0\rangle.$$

Analogamente, si hemos obtenido el valor 1, entonces:

$$|\psi'\rangle = \frac{M_1|\psi\rangle}{\sqrt{\langle\psi|M_1^\dagger M_1|\psi\rangle}} = \frac{\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}}{\sqrt{\begin{bmatrix} a & b \end{bmatrix}\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}}} = \frac{b}{|b|}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{b}{|b|}|1\rangle.$$

Como ya hemos mencionado, los posibles estados de un sistema cuántico son infinitos, sin embargo, no todos ellos nos resultarán útiles, puesto que solo podremos *distinguir* entre sí aquellos que sean ortogonales. Para explicar mejor esta idea, cojamos dos estados cualquiera $|\psi_1\rangle$ y $|\psi_2\rangle$. Supongamos que no son ortogonales. Esto tiene como consecuencia que $|\psi_2\rangle$ pueda descomponerse como $a|\psi_1\rangle + b|\phi\rangle$, siendo $|\phi\rangle$ un vector unitario ortogonal a $|\psi_1\rangle$.

Esto es debido a que $|\psi_1\rangle$ y $|\phi\rangle$ formarían una base ortonormal. Supongamos que elegimos un conjunto de operadores de medida $\{M_m\}$, de manera que se obtiene el resultado m_i si el sistema está en el estado $|\psi_i\rangle$. Ahora, preparemos el sistema para que se encuentre en el estado $|\psi_2\rangle$. Al aplicar la medición, esta debería devolver el resultado m_2 con toda seguridad. Sin embargo, como $|\psi_2\rangle$ contiene una componente paralela a $|\psi_1\rangle$, la probabilidad de obtener el resultado m_1 es no nula, produciendo una medición errónea y concluyendo que los estados no se pueden distinguir de forma fiable. Es por eso que estos estados se denominan *indistinguibles*. Por ejemplo, tomemos las condiciones del ejemplo 3.2. Sean los estados $|0\rangle$ y $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. De este modo, al medir el primer estado siempre obtenemos 0, mientras que en el segundo puede darse 1 o 0, por lo que si al medir obtenemos un 0, no podemos saber el estado en el que se encontraba el sistema.

Ahora introduciremos un tipo especial de operadores de medida denominados *medidas proyectivas*. Este enfoque para la medición nos dará ventajas prácticas, puesto que nos da una forma de establecer los resultados de una observación.

Definición 3.1. Definimos *observable* como un operador hermitiano que tiene como espacio de salida el espacio de estados del sistema físico que vamos a medir.

En la práctica, un observable es cualquier magnitud física que se puede medir en un sistema cuántico.

Usamos M indistintamente para el operador como aplicación y como su matriz asociada en la base usual. Al ser M hermitiana, es normal, y sabemos que es posible realizar su descomposición espectral en proyectores (1.3) a partir de los autovalores de M.

$$M = \sum_{\lambda} \lambda P_\lambda.$$

En esta ecuación, λ es un autovalor de M y P_λ es un proyector sobre el subespacio generado por el autovector asociado a dicho valor. Adaptando las fórmulas del postulado 3.4 obtenemos que si un sistema está en un estado $|\psi\rangle$, los resultados de la medición serán los autovalores de M, y la probabilidad de obtener un autovalor λ concreto vendrá dada por:

$$p(\lambda) = \langle\psi| P_\lambda |\psi\rangle .$$

Por otro lado, si el resultado λ_0 es el que se ha obtenido, entonces el estado del sistema tras realizar la medición será:

$$|\psi'\rangle = \frac{P_{\lambda_0}|\psi\rangle}{\sqrt{p(\lambda_0)}}.$$

Una posible forma de medir es tomar una base ortonormal $B = \{|1\rangle, \ldots, |n\rangle\}$ y usando los operadores de medición $P_i = |i\rangle\langle i|$ con $i = 1, \ldots, n$. A este proceso se le denomina *medir en la base B*.

Ejemplo 3.3. Si tomamos de nuevo las condiciones del ejemplo 3.2, es fácil de ver que si definimos el observable:

$$M = 0M_0^\dagger M_0 + 1M_1^\dagger M_1 = 0\,|0\rangle\langle 0| + 1\,|1\rangle\langle 1| = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix},$$

entonces las formulas para la probabilidad y estado después de medición son equivalentes a las definidas en el postulado 3.4. Además, realizar dicha medición sería "medir en la base $|0\rangle$,$|1\rangle$", que podría arrojar como valores posibles 0 y 1.

Otra posibilidad es usar las llamadas *matrices de Pauli*, que son matrices 2×2 que tienen la siguiente forma:

$$\sigma_1 = \sigma_x = X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$
$$\sigma_2 = \sigma_y = Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix},$$
$$\sigma_3 = \sigma_z = Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

A partir de estas matrices, podemos definir un observable. Sea $v = (v_1, v_2, v_3) \in \mathbb{R}^3$. Entonces definimos:

$$\vec{v} \cdot \vec{\sigma} = v_1\sigma_1 + v_2\sigma_2 + v_3\sigma_3.$$

El utilizar este observable para la medición se le denomina *medida del spin a lo largo del eje* $\vec{v}$ por motivos historicos.

Una vez que hemos establecido la forma en la que medimos un sistema, pasaremos al último postulado, en el que se describe el comportamiento de lo que llamaremos *sistemas compuestos*, que no son más que sistemas formados por otros más simples. La idea principal es que podremos utilizar la composición de la información disponible sobre los sistemas más simples para describir aquel formado por estos.

Postulado 3.5. *Un sistema compuesto viene dado por el producto tensorial de los sistemas que lo componen. De forma más concreta, si tenemos n sistemas preparados en los estados* $|\psi_i\rangle$ *con* $i = 1 \dots n$*, entonces el sistema compuesto formado por estos queda descrito por el estado* $|\psi_1\rangle \otimes \cdots \otimes |\psi_n\rangle$.

En nuestro caso, el producto tensorial que utilizaremos será el productor de Kronecker (1.2), que definimos en el capítulo 1. Para poner de manifiesto esta idea, veamos un ejemplo.

Ejemplo 3.4. Tomamos dos qubits, que pueden describir un sistema de dos partículas con dos estados, $|0\rangle$ y $|1\rangle$. Tomemos la misma base para ambos sistemas, que puede ser cualquier base ortonormal. Los estados de estos dos sistemas son:

$$|\psi_1\rangle = a_1 |0\rangle + b_1 |1\rangle,$$
$$|\psi_2\rangle = a_2 |0\rangle + b_2 |1\rangle.$$

Entonces, por el postulado anterior, podemos definir el estado del sistema formado por estos dos qubits como:

$$\begin{aligned} |\psi_1\rangle \otimes |\psi_2\rangle = (a_1 |0\rangle + b_1 |1\rangle) \otimes (a_2 |0\rangle + b_2 |1\rangle) = \\ a_1a_2(|0\rangle \otimes |0\rangle) + a_1b_2(|0\rangle \otimes |1\rangle) + b_1a_2(|1\rangle \otimes |0\rangle) + b_1b_2(|1\rangle \otimes |1\rangle) = \\ a_1a_2 |00\rangle + a_1b_2 |01\rangle + b_1a_2 |10\rangle + b_1b_2 |11\rangle. \end{aligned}$$

Esta es una nueva combinación lineal de vectores en el espacio producido como producto de Kronecker de los espacios de estado en el que estaban $|\psi_1\rangle$ y $|\psi_2\rangle$. Para poder calcular cada uno de los $|ij\rangle$, será necesario fijar una base. Como ejemplo, cojamos $B = \{|0\rangle, |1\rangle\}$ con $|0\rangle = \left[1,0\right]^T$ y $|1\rangle = \left[0,1\right]^T$. Llamamos a esta la base la *base computacional*.

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{bmatrix} 1\begin{bmatrix}1\\0\end{bmatrix} \\ 0\begin{bmatrix}1\\0\end{bmatrix} \end{bmatrix} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, |01\rangle = |0\rangle \otimes |1\rangle = \begin{bmatrix} 1\begin{bmatrix}0\\1\end{bmatrix} \\ 0\begin{bmatrix}0\\1\end{bmatrix} \end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix},$$

$$|10\rangle = |1\rangle \otimes |0\rangle = \begin{bmatrix} 0\begin{bmatrix}1\\0\end{bmatrix} \\ 1\begin{bmatrix}1\\0\end{bmatrix} \end{bmatrix} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, |11\rangle = |1\rangle \otimes |1\rangle = \begin{bmatrix} 0\begin{bmatrix}0\\1\end{bmatrix} \\ 1\begin{bmatrix}0\\1\end{bmatrix} \end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}.$$

De modo que la combinación lineal final es:

$$|\psi_1\rangle \otimes |\psi_2\rangle = \begin{bmatrix} a_1a_2 \\ a_1b_2 \\ b_1a_2 \\ b_1b_2 \end{bmatrix}.$$

Además, se cumple que el vector resultante sigue siendo unitario.

$$\begin{aligned} \||\psi_1\rangle \otimes |\psi_2\rangle\| &= \sqrt{|a_1a_2|^2 + |a_1b_2|^2 + |b_1a_2|^2 + |b_1b_2|^2} = \\ &\sqrt{|a_1|^2(|a_2|^2 + |b_2|^2) + |b_1|^2(|a_2|^2 + |b_2|^2)} = \\ &\sqrt{|a_1|^2 + |b_1|^2} = 1. \end{aligned}$$

De esta manera, podemos ver que el vector producido por $|\psi_1\rangle \otimes |\psi_2\rangle$ es un vector de estado válido.

Gracias a estos postulados tenemos las herramientas más básicas que sentarán las reglas para la manipulación de sistemas cuánticos.

3.3. El principio de indeterminación de Heisenberg

La naturaleza probabilística de los resultados de una medición nos permite tratarlos como si fueran las imágenes de una variable aleatoria. La forma alternativa de medición por observables permite calcular la media ponderada de los valores posibles, también llamada esperanza matemática, de manera sencilla. El cálculo de dicha media es el siguiente:

$$E(M) = \sum_\lambda \lambda p(\lambda) = \sum_\lambda \lambda \langle\psi| P_\lambda |\psi\rangle = \langle\psi| \sum_\lambda \lambda P_\lambda |\psi\rangle = \langle\psi| M |\psi\rangle.$$

En general, la esperanza matemática o valor medio de un observable se escribirá como $\langle M\rangle$. Una vez que tenemos el valor medio de un observable, es posible calcular su *desviación*

típica:

$$\Delta(M) = \sqrt{\langle (M - I\langle M\rangle)^2\rangle}.$$

A continuación presentaremos el principio de incertidumbre de Heisenberg. Este principio representa un cambio radical en la forma en que entendemos la realidad. Hasta el momento en que fue propuesto, se creía que la física clásica podía proporcionar una descripción completa y precisa de todos los fenómenos físicos. Sin embargo, el principio de incertidumbre muestra que en el mundo subatómico, las cosas son muy diferentes. La incertidumbre es una característica fundamental de la naturaleza y el principio de incertidumbre nos obliga a reconsiderar nuestras intuiciones sobre cómo funciona el universo.

Teorema 3.1. *Supongamos un sistema cuántico con V su espacio de estados, $|\psi\rangle$ un estado en este y A y B dos observables sobre V. Entonces se cumple la siguiente desigualdad:*

$$\Delta(A)\Delta(B) \geq \frac{|\langle\psi|[A,B]|\psi\rangle|}{2}. \tag{3.4}$$

La interpretación de este teorema es que si preparamos un gran número de sistemas cuánticos en estados idénticos, $|\psi\rangle$, y luego realizamos mediciones de A en algunos de esos sistemas, y de B en otros, entonces la desviación típica $\Delta(A)$ de los resultados de A multiplicada por la desviación típica $\Delta(B)$ de los resultados de B satisfará la desigualdad (3.4).

3.4. Teorema de la no clonación

Los circuitos cuánticos no son más que modelos que utilizamos para representar operaciones sobre un sistema cuántico. Estos circuitos se basan en la misma idea que sus equivalentes convencionales presentados en el capítulo 2. Introducimos los circuitos cuánticos debido a que son importantes para entender los fenómenos de entrelazamiento y teletransporte cuánticos. De la misma manera que en los circuitos convencionales, los circuitos se construyen a partir de cables, cada uno representando un qubit, pasando por puertas lógicas que son operaciones sobre dichos qubits. Una vez pasemos al aspecto práctico, a menos que se especifique lo contrario, siempre utilizaremos la base computacional, cuyos estados $|0\rangle$ y $|1\rangle$ se corresponden los estados de un bit clásico.

Ejemplo 3.5. Un circuito extremadamente simple es el de negación , usando la puerta NOT, que se representa mediante la matriz de Pauli $\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.

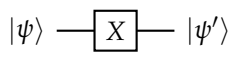


Figura 3.2: Circuito que niega la entrada.

Para calcular el resultado de este circuito solo tenemos que aplicar el operador $X = \sigma_x$ al

vector de entrada. De manera que tenemos:

$$X|\psi\rangle = X(a|0\rangle + b|1\rangle) = X\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} b \\ a \end{bmatrix} = |\psi'\rangle .$$

En el ejemplo anterior se hace uso de un sistema con un solo qubit. La puerta CNOT es la más simple dentro de aquellas que se aplican a un sistema de dos qubits. Esta puerta funciona exactamente igual que con los bits clásicos. El circuito tiene la siguiente forma:

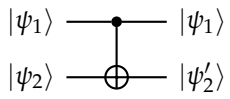


Figura 3.3: Circuito que niega la entrada dependiendo de la línea de control.

En este caso aplicamos una transformación que tiene la forma:

$$|00\rangle \to |00\rangle , |01\rangle \to |01\rangle , |10\rangle \to |11\rangle , |11\rangle \to |10\rangle .$$

La forma en la que calculamos la matriz asociada a esta transformación será discutirá con mucho más detalle en el próximo capítulo.

Como podemos ver en el ejemplo anterior, la puerta CNOT nos puede servir para copiar bits clásicos: cuando la entrada de la segunda línea se fija a $|0\rangle$, el resultado es lo que sea que haya en la primera línea, $|0\rangle$ o $|1\rangle$. Sin embargo, esto deja de ser así cuando en la primera línea se encuentra un estado cualquiera $|\psi_1\rangle = a|0\rangle + b|1\rangle$. Veamos por qué.

La entrada del circuito 3.4 cuanto se dan estas condiciones es:

$$|\psi_1\rangle|0\rangle = (a|0\rangle + b|1\rangle)|0\rangle = a|00\rangle + b|10\rangle .$$

Las puertas se aplican linealmente, ya que en definitiva todas son operadores lineales. De manera que el resultado de aplicar la puerta CNOT a esta entrada es:

$$\text{CNOT}(|\psi_1\rangle|0\rangle) = \text{CNOT}(a|00\rangle + b|10\rangle) = a|00\rangle + b|11\rangle .$$

De este modo, vemos que la salida es una superposición de estados. Por otra parte, si lo que queremos es copiar el qubit $|\psi_1\rangle$, entonces la salida del circuito debería ser:

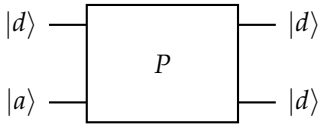
$$|\psi_1\rangle|\psi_1\rangle = a^2|00\rangle + ab|01\rangle + ab|10\rangle + b^2|11\rangle .$$

Como vemos, la salida esperada solo es igual a la que obtenemos si $ab = 0$ y $a = a^2$ y $b = b^2$, pero esto implicaría que $a = 0$ y $b = 1$ o $b = 1$ y $a = 0$, con lo que tendríamos que $|\psi_1\rangle = |0\rangle$ o $|\psi_1\rangle = |1\rangle$.

La conclusión que obtenemos es que no es posible copiar un estado cualquiera desconocido distinto de aquellos de la base computacional con la puerta CNOT. Pero, ¿qué ahí de otros circuitos o combinación de puertas? La respuesta sigue siendo negativa. Esta conclusión se

denomina *teorema de la no clonación*.

Supongamos que podemos crear un circuito con dos entradas, con valores $|d\rangle$ y $|a\rangle$ (línea de datos y línea de almacenamiento), al que le queremos aplicar una determinada transformación unitaria P tal que la salida es $|d\rangle$ en ambas líneas, es decir:



Una transformación tal que $P(|d\rangle \otimes |a\rangle) = |d\rangle \otimes |d\rangle$. Además, supongamos que esta transformación cumple la igualdad anterior para dos estados distintos:

$$P(|d_1\rangle \otimes |a\rangle) = |d_1\rangle \otimes |d_1\rangle\,,$$
$$P(|d_2\rangle \otimes |a\rangle) = |d_2\rangle \otimes |d_2\rangle\,.$$

Podemos tomar el producto escalar de ambas ecuaciones, de modo que tenemos:

$$\langle P(|d_1\rangle \otimes |a\rangle), P(|d_2\rangle \otimes |a\rangle)\rangle = \langle d_1|\, P^\dagger P\, |d_2\rangle = \langle d_1|d_2\rangle\,,$$
$$\langle |d_1\rangle \otimes |d_1\rangle\,, |d_2\rangle \otimes |d_2\rangle\rangle = \langle d_1|d_2\rangle^2\,.$$

Al final tenemos que $\langle d_1|d_2\rangle = \langle d_1|d_2\rangle^2$, lo cual solo se cumple si $\langle d_1|d_2\rangle = 1$ o $\langle d_1|d_2\rangle = 0$, de modo que o bien $|d_1\rangle = |d_2\rangle$ o bien $|d_1\rangle \perp |d_2\rangle$. Esto implica que un hipotético dispositivo de clonación solo funcionaría para estados que sean perpendiculares entre sí. Es por esto que en el ejemplo de la figura 3.4 el dispositivo funcionaba para los estados $|d_1\rangle = |0\rangle$ y $|d_2\rangle = |1\rangle$, ya que son perpendiculares. Sin embargo, un dispositivo general es imposible de construir.

3.5. Entrelazamiento cuántico

El entrelazamiento cuántico es uno de los fenómenos más fascinantes y misteriosos de la física cuántica. Como ya hemos mencionado, al contrario que los objetos clásicos que existen en un estado definido, los sistemas cuánticos pueden existir en múltiples estados simultáneamente, y estos estados pueden estar entrelazados entre sí. El entrelazamiento cuántico describe la correlación entre dos o más partículas cuánticas que comparten un estado cuántico. Esto significa que si una partícula cambia su estado, la otra partícula cambia instantáneamente su estado también, independientemente de la distancia que las separe.

Para introducir el concepto de entrelazamiento, presentaremos primero una puerta cuántica de vital importancia: la *puerta de Hadamard*. Esta transformación se describe como:

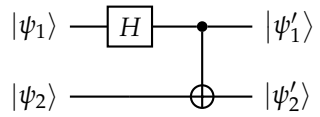
$$H\,|\psi\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}a\\ b\end{bmatrix} = \begin{bmatrix}\frac{a}{\sqrt{2}} + \frac{b}{\sqrt{2}}\\ \frac{a}{\sqrt{2}} - \frac{b}{\sqrt{2}}\end{bmatrix}$$

Esta puerta es de gran utilidad porque permite transformar los estados fundamentales $|0\rangle$

y $|1\rangle$ en una superposición de los mismos:

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

A continuación, utilizamos esta puerta para construir el siguiente circuito:



Si limitamos las entradas de este circuito a $|\psi_i\rangle = |j\rangle$ con $i, j = 1, 2$, la transformación que ocurre primero es la puerta de Hadamard, pone a $|\psi_1\rangle$ en superposición. Este qubit sirve después como control de la puerta CNOT. Sin embargo, ¿cómo funcionaría esto? La puerta CNOT se basa en cambiar la puerta objetivo de $|0\rangle$ a $|1\rangle$ y viceversa en función del qubit de control, pero si el qubit de control está en una superposición de estados, ¿qué ocurre? Vayamos paso a paso en el circuito anterior.

Tras aplicar la puerta de Hadamard al primer qubit, las posibilidades para los resultados son:

$$|00\rangle \to (H|0\rangle) \otimes |0\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle,$$
$$|01\rangle \to (H|0\rangle) \otimes |1\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|11\rangle,$$
$$|10\rangle \to (H|1\rangle) \otimes |0\rangle = \frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|10\rangle,$$
$$|11\rangle \to (H|1\rangle) \otimes |1\rangle = \frac{1}{\sqrt{2}}|01\rangle - \frac{1}{\sqrt{2}}|11\rangle.$$

Ahora, aplicamos la puerta CNOT linealmente al estado que forma la superposición, obteniendo la transformación del circuito completo:

$$|00\rangle \to \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle,$$
$$|01\rangle \to \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle,$$
$$|10\rangle \to \frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|11\rangle,$$
$$|11\rangle \to \frac{1}{\sqrt{2}}|01\rangle - \frac{1}{\sqrt{2}}|10\rangle.$$

Estos estados resultantes, debido a su importancia histórica, reciben el nombre *estados de Bell*, *estados EPR* o *parejas EPR*, donde las siglas EPR provienen de los físicos que trabajaron con

ellos: Einstein, Podolsky y Rosen.

Tomamos, por ejemplo, $|00\rangle$ como entrada. El resultado del circuito es $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$. Al realizar una medición en la base computacional tenemos que las posibilidades son claramente $|00\rangle$ y $|11\rangle$. Esto nos dice que si midiéramos un solo qubit, el resultado del otro quedaría totalmente determinado. Es decir, existe una *dependencia* entre ambos qubits, de manera que al medir uno, podemos conocer el resultado de ambos. Es natural que esta dependencia exista, puesto que si el qubit de control hubiese sido $|1\rangle$, entonces el segundo qubit se habría visto modificado. Además, como el primer qubit ya se ha visto determinado a partir del segundo, el postulado 3.4 nos dice que este ha cambiado para acomodar dicha medición. Esto revela un fenómeno extraño: hemos hecho cambiar un qubit sin interactuar con él. Estos qubits se dice que están *entrelazados*. Más formalmente:

Definición 3.2. Se dice que un sistema físico formado por dos partículas en un estado $|\psi\rangle$ son *qubits entrelazados* cuando no existen, dentro del espacio de estados, vectores $|\psi_1\rangle$ y $|\psi_2\rangle$ tales que $|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$.

Ejemplo 3.6. Veamos que los estados de Bell son sistemas de dos qubits entrelazados. Supongamos que existe $|\psi_1\rangle$ y $|\psi_2\rangle$ tales que $|\psi_1\rangle \otimes |\psi_2\rangle = a|00\rangle + b|11\rangle$. Si $|\psi_1\rangle = a_1|0\rangle + b_1|1\rangle$ y $|\psi_2\rangle = a_2|0\rangle + b_2|1\rangle$, entonces tenemos que:

$$a|00\rangle + b|11\rangle = (a_1|0\rangle + b_1|1\rangle) \otimes (a_2|0\rangle + b_2|1\rangle) = a_1a_2|00\rangle + b_1a_2|10\rangle + a_1b_2|01\rangle + b_1b_2|11\rangle .$$

Al ser todos estos vectores linealmente independientes, tenemos que $b_1a_2 = 0 = a_1b_2$, lo que nos lleva claramente a contradicción. Se puede aplicar el mismo proceso al resto de los estados de Bell, demostrando así que son producto del entrelazamiento de dos qubits.

Exploraremos con profundidad las múltiples aplicaciones de este fenómeno en el próximo capítulo.

3.6. Operador densidad

En esta sección introduciremos una formulación alternativa a los vectores de estado y en general a las propiedades de la mecánica cuántica definidos en la sección 3.2. Una formulación Una alternativa es posible utilizando una herramienta conocida como *operador densidad* o *matriz densidad*. Esta formulación alternativa es matemáticamente equivalente al enfoque de vectores de estado, pero proporciona un lenguaje mucho más conveniente para pensar en algunos escenarios comúnmente en la mecánica cuántica. Una de estas posibilidades es la del manejo de estados cuánticos que no son del todo conocidos o "estados mezcla". Por otra parte, este operador también nos ofrecerá facilidades para obtener información de los sistemas cuánticos que componen un sistema complejo.

Supongamos que tenemos un sistema cuántico que puede estar en unos ciertos estados $|\psi_i\rangle$ conocidos cada uno con una probabilidad p_i, con $i \in I$ un conjunto de índices. El conjunto de las parejas $\{(p_i, |\psi_i\rangle)\}$ se denomina *configuración de estados puros*. A partir de ellos definimos la matriz densidad de un sistema como:

$$\rho \equiv \sum_{i \in I} p_i |\psi_i\rangle \langle\psi_i|.$$

3.6.1. Redefinición de los postulados de la mecánica cuántica

Como ya hemos mencionado, podemos reformular las reglas definidas por los postulados de la sección 3.2 mediante esta nomenclatura. En primer lugar, vamos a definir cómo evoluciona un sistema en función de su matriz densidad. Supongamos una transformación unitaria descrita por una matriz U. Entonces, como vimos en el postulado 3.2, si el sistema podía presentarse en una serie de posibles estados $|\psi\rangle$ con probabilidades p_i, los nuevos posibles estados del sistema son $|\psi_i'\rangle = U|\psi_i\rangle$ con las mismas probabilidades. Por ende, la matriz densidad del sistema tras la transformación será:

$$\rho' = \sum_i p_i |\psi_i'\rangle \langle\psi_i'| = \sum_i p_i U |\psi_i\rangle \langle\psi_i| U^\dagger = U\rho U^\dagger.$$

Por otro lado, la medición también puede describirse fácilmente con este operador. Tomemos un conjunto de operadores de medición $\{M_m\}$. La probabilidad de obtener el resultado m condicionada a que el sistema se encuentre en el estado $|\psi_i\rangle$ se calcula como:

$$p(m|i) = \langle\psi_i| M_m^\dagger M_m |\psi_i\rangle = tr(M_m^\dagger M_m |\psi_i\rangle \langle\psi_i|), \tag{3.5}$$

donde hemos utilizado la propiedad (1.1). Sin embargo, la ecuación que acabamos de ver solo nos da la probabilidad condicionada. Esto no es problema, ya que podemos utilizar la ley de la probabilidad total. Si tomamos $p(i) \equiv p_i$ como la probabilidad de que el sistema esté en el estado con índice i:

$$p(m) = \sum_i p(m|i)p(i) = \sum_i p_i tr(M_m^\dagger M_m |\psi_i\rangle \langle\psi_i|) = tr(M_m^\dagger M_m \rho).$$

Además, el estado del sistema tras la medición, si se ha obtenido el resultado m, se determina a partir de:

$$|\psi^m\rangle = \frac{M_m |\psi_i\rangle}{\sqrt{\langle\psi_i| M_m^\dagger M_m |\psi_i\rangle}}.$$

Después de haber realizado la medición, obtendremos una configuración de estados $|\psi^m\rangle$, con sus respectivas probabilidades $p(i|m)$, esto es, la probabilidad de obtener el estado de índice i condicionada a haber obtenido el resultado m. Por tanto, la matriz densidad del nuevo sistema es:

$$\rho_m = \sum_i p(i|m) |\psi^m\rangle \langle\psi^m| = \sum_i p(i|m) \frac{M_m |\psi_i\rangle \langle\psi_i| M_m^\dagger}{\langle\psi_i| M_m^\dagger M_m |\psi_i\rangle}, \tag{3.6}$$

pero, por definición de probabilidad condicionada, tenemos:

$$p(i|m) = \frac{p(i,m)}{p(m)} = \frac{p(m|i)p_i}{p(m)}. \tag{3.7}$$

Así que podemos sustituir (3.5) y (3.7) en (3.6), obteniendo:

$$\rho_m = \sum_i p_i \frac{M_m |\psi_i\rangle \langle\psi_i| M_m^\dagger}{tr(M_m^\dagger M_m \rho)} = \frac{M_m \rho M_m^\dagger}{tr(M_m^\dagger M_m \rho)}.$$

Además, ahora podemos redefinir la esperanza a partir del operador densidad:

$$E(M) = \sum_m mp(m) = \sum_m mtr(M_m^\dagger M_m \rho) = tr(\sum_m mM_m^\dagger M_m \rho) = tr(M\rho)$$

Para terminar de reescribir los postulados fundamentales en el lenguaje del operador densidad, tomemos un conjunto de n sistemas cuánticos de una sola partícula, con sus configuraciones de estados correspondientes $\{(p_{i1}, |\psi_{i_1 1}\rangle)\}, \ldots, \{(p_{in}, |\psi_{in}\rangle)\}$. Cada configuración tiene un conjunto de índices I_j. Si hacemos el producto cartesiano $I = I_1 \times \cdots \times I_n$, acabamos con un conjunto de tuplas de índices representando todas las combinaciones posibles de estados. ¿Cuál es la probabilidad de encontrar el sistema s_j en el estado $|\psi_{ij}\rangle$, con $j = 1, \ldots, n$? Claramente es $p_{(i_1,\ldots,i_n)} = \prod_{k=1}^n p_{i_k}$. Haciendo uso del postulado 3.5, tomamos:

$$\left|\psi_{(i_1,\ldots,i_n)}\right\rangle := |\psi_{i_1 1}\rangle \otimes \cdots \otimes |\psi_{i_n n}\rangle .$$

Ahora podemos calcular la matriz densidad del sistema compuesto, obteniendo:

$$\rho = \sum_{(i_1,\ldots,i_n)\in I} \left(p_{(i_1,\ldots,i_n)} \left|\psi_{(i_1,\ldots,i_n)}\right\rangle \left\langle\psi_{(i_1,\ldots,i_n)}\right| \right) = \sum_{i_1 \in I_1} p_{i_1} |\psi_{i_1 1}\rangle \langle\psi_{i_1 1}| \otimes \cdots \otimes \sum_{i_n \in I_n} p_{i_n} |\psi_{i_n n}\rangle \langle\psi_{i_n n}|$$

$$= \rho_1 \otimes \cdots \otimes \rho_2.$$

Siendo ρ_j la matriz densidad del sistema j.

De esta manera, hemos podido reescribir el postulado 3.5 en el lenguaje de la matriz densidad, llegando a la conclusión de que un sistema compuesto queda totalmente descrito por el producto de Kronecker de las matrices densidad de los sistemas que lo componen.

3.6.2. Estados puros y estados mezcla

A continuación introduciremos un nuevo concepto que tiene como base la naturaleza probabilística del lenguaje del operador densidad.

Definición 3.3. Dado un sistema cuántico cualquiera se dice que está en un *estado puro* cuando se encuentra en un estado totalmente conocido $|\psi\rangle$. En este caso, el operador densidad es $\rho = |\psi\rangle \langle\psi|$. En caso contrario, se dice que el sistema está en un *estado mezcla*.

La siguiente proposición nos dará un criterio muy útil para saber de antemano si una matriz densidad representa un sistema en un estado puro o no.

Proposición 3.1. *Dado un sistema cuántico con una matriz densidad ρ, se tiene que $tr(\rho^2) \leq 1$ y la igualdad se cumple si y solo si el sistema se encuentra en un estado puro.*

Demostración. Por definición tenemos que, para una determinada configuración de estados cuánticos, $\{(p_i, |\psi_i\rangle)\}$, la matriz densidad del sistema es:

$$\rho = \sum_i p_i |\psi_i\rangle \langle\psi_i|.$$

Tenemos que si $|\psi_i\rangle = \left[a_1^i, \ldots, a_n^i\right]^T$ entonces:

$$tr(|\psi_i\rangle \langle\psi_i|) = \sum_{j=1}^{n} (a_j^i)^2 = \langle\psi_i, \psi_i\rangle = 1,$$

para todo i. Luego:

$$tr(\rho) = \sum_i tr(p_i |\psi_i\rangle \langle\psi_i|) = \sum_i p_i = 1.$$

Es fácil ver que esta matriz es hermitiana, puesto que:

$$\rho^\dagger = (\sum_i p_i |\psi_i\rangle \langle\psi_i|)^\dagger = \sum_i (p_i |\psi_i\rangle \langle\psi_i|)^\dagger = \sum_i p_i |\psi_i\rangle \langle\psi_i| = \rho.$$

Donde hemos usado que p_i es un número real y que $|\psi_i\rangle^\dagger = \langle\psi_i|$. Además, sabemos que ρ es semidefinida positiva:

$$\langle\varphi| \rho |\varphi\rangle = \langle\varphi| (\sum_i p_i |\psi_i\rangle \langle\psi_i|) |\varphi\rangle = \sum_i p_i \langle\varphi|\psi_i\rangle^2 \geq 0, \forall |\varphi\rangle .$$

A continuación, podemos realizar la descomposición espectral de dicha matriz. Sabemos que existe una base ortonormal $B = \{|\Phi_1\rangle, \ldots, |\Phi_n\rangle\}$ y una matriz unitaria U tal que:

$$\rho = U(\sum_i \lambda_i |\Phi_i\rangle \langle\Phi_i|)U^\dagger \implies \rho^2 = U(\sum_i \lambda_i^2 |\Phi_i\rangle \langle\Phi_i|)U^\dagger.$$

Tal que $\lambda_i \geq 0$. Pero $tr(\rho) = 1$, luego:

$$tr(\rho) = tr(U(\sum_i \lambda_i |\Phi_i\rangle \langle\Phi_i|)U^\dagger) = \sum_i^n \lambda_i = 1 \implies$$
$$tr(\rho^2) = tr(U(\sum_i \lambda_i^2 |\Phi_i\rangle \langle\Phi_i|)U^\dagger) = \sum_i^n \lambda_i^2 \leq 1.$$

Ya hemos probado que $tr(\rho^2) \leq 1$. Vayamos ahora a probar que ρ está en un estado puro si y solo si se da la igualdad.

$\underline{\Longrightarrow}$

Supongamos ahora que ρ está en un estado puro. Es decir, $\rho = |\psi_0\rangle \langle\psi_0|$. Entonces tenemos que:

$$\rho^2 = |\psi_0\rangle \langle\psi_0|\psi_0\rangle \langle\psi_0| = |\psi_0\rangle \langle\psi_0| ,$$

de lo que se deduce fácilmente que $tr(\rho^2) = 1$.

$\underline{\Longleftarrow}$

Para probar la implicación contraria, suponemos que $tr(\rho^2) = 1$. Pero $tr(\rho) = \sum_i^n \lambda_i = 1$, de modo que $\lambda_i \leq 1 \ \forall i$. Si $\lambda_i < 1 \ \forall i$, entonces claramente $\sum_i^n \lambda_i^2 < 1$. Por tanto, debe exister i tal que $\lambda_i = 1$ y $\lambda_j = 0$ si $j \neq i$. Luego si tomamos la descomposición espectral que hemos usado en el apartado anterior, podemos escribir ρ como:

$$\rho = U(\sum_i \lambda_i |\Phi_i\rangle \langle\Phi_i|)U^\dagger,$$

pero teniendo en cuenta que solo uno de los λ_i es distinto de 0, nos queda:

$$\rho = U |\Phi_i\rangle \langle\Phi_i| U^\dagger.$$

Si renombramos $|\phi\rangle = U |\Phi\rangle$, entonces acabamos con la definición de estado puro.

□

3.6.3. Caracterización

Teorema 3.2. *Un operador ρ es la matriz densidad correspondiente a una configuración de estados cuánticos $\{(p_i, |\psi_i\rangle)\}$ si y solo si $tr(\rho) = 1$ y ρ es un operador semidefinido positivo.*

Demostración. $\Longrightarrow$

Supongamos que ρ es una matriz densidad, luego:

$$\rho = \sum_i p_i |\psi\rangle \langle\psi| \implies$$
$$tr(\rho) = \sum_i p_i tr(|\psi_i\rangle \langle\psi_i|) = \sum_i p_i = 1,$$
$$\langle\varphi| \rho |\varphi\rangle = \langle\varphi| (\sum_i p_i |\psi_i\rangle \langle\psi_i|) |\varphi\rangle = \sum_i p_i \langle\varphi|\psi_i\rangle^2 \geq 0, \forall |\varphi\rangle .$$

$\Longleftarrow$

Supongamos ρ un operador cuya traza es 1 y que es semidefinido positivo. Al ser semidefinido positivo, podemos realizar su descomposición espectral:

$$\rho = \sum_{i=1}^{n} \lambda_i |i\rangle \langle i|.$$

Siendo $B = \{|1\rangle, \ldots, |n\rangle\}$ una base ortonormal y $\{\lambda_1, \ldots, \lambda_n\}$ el espectro de ρ. Además, es fácil ver que todos los autovalores son positivos. Como $0 < \lambda_i \leq 1$ para $i = 1, \ldots, n$, podemos tratar estos autovalores como imágenes de una función de probabilidad, de forma que concluimos que ρ es el operador densidad para una configuración de estados cuánticos $\{(\lambda_i, |i\rangle)\}$.

□

3.6.4. Operador densidad reducido

Al principio de esta sección mencionamos como uno de los motivos principales para introducir el lenguaje del operador densidad es estudiar los subsistemas cuánticos que componen un sistema más grande. Para llevar esto acabo, haremos uso de un nuevo concepto: el *operador densidad reducido*.

Sean dos sistemas cuánticos s_1 y s_2, y el sistema completo $s_1 \otimes s_2$. Ahora, definiremos dos observables: M, que mide una propiedad del sistema s_1 y $\tilde{M}$, otro observable que mide la misma propiedad en el sistema compuesto $s_1 \otimes s_2$. Ahora, si P_m es el proyector al autoespacio asociado al autovalor m en el espacio de estados de s_1, entonces el proyector correspondiente a $\tilde{M}$ en el espacio de estados de $s_1 \otimes s_2$ será $P_m \otimes I_{s_2}$, donde I_{s_2} es el operador indetidad en

s_2, ya que $\tilde{M}$ mide la misma propiedad. Por ende, debe cumplirse que:

$$\tilde{M} = \sum_m m P_m \otimes I_{s_2} = M \otimes I_{s_2}.$$

Por otra parte, imaginemos que existe un operador densidad, $\rho_{s_1'}$, calculado a partir de la matriz densidad del sistema compuesto, $\rho_{s_1 s_2}$. La medición del observable M en s_1 debe ser consistente con la medición de $\tilde{M}$ en $s_1 \otimes s_2$, por lo tanto, la esperanza debe mantenerse:

$$tr(M\rho_{s_1'}) = tr(\tilde{M}\rho_{s_1 s_2}) = tr((M \otimes I_{s_2})\rho_{s_1 s_2}). \tag{3.8}$$

El cálculo que necesitamos hacer para calcular dicho operador $\rho_{s_1'}$ se denomina *traza parcial*. Si $|v_{i1}\rangle$, $|v_{i2}\rangle$ son vectores del espacio de estados de s_i con $i = 1, 2$ de dimensiones cualesquiera, entonces la traza parcial se define como:

$$tr_{s_2}(|v_{11}\rangle \langle v_{12}| \otimes |v_{21}\rangle \langle v_{22}|) = |v_{11}\rangle \langle v_{12}| \, tr(|v_{21}\rangle \langle v_{22}|) = |v_{11}\rangle \langle v_{12}| \langle v_{22}|v_{21}\rangle .$$

El operador que cumple (3.8) se calcula aplicando la traza parcial a la matriz densidad del tema compuesto:

$$\rho_{s_1} := tr_{s_2}(\rho_{s_1 s_2}).$$

El operador densidad resultante, ρ_{s_1}, se llama *operador densidad reducido*.

Este operador describe totalmente el sistema s_1. Dicho coloquialmente, esta operación es "contraria" al producto de Kronecker en este contexto, puesto que nos permite conocer las características de sistemas individuales a partir de la matriz densidad del sistema completo. Esto puede comprobarse suponiendo dos sistemas con matrices de densidad ρ_i, $i = 1, 2$ y el sistema $\rho_1 \otimes \rho_2$. Si hacemos la traza parcial del primer sistema, $tr_{s_2}(\rho_1 \otimes \rho_2) = \rho_1$, que como queríamos, nos da la matriz densidad del primer sistema. Veamos un ejemplo algo más práctico y mucho más famoso.

Ejemplo 3.7 (Gato de Schrödinger)**.** El gato de Schrödinger es un experimento puramente mental que se creó con el fin de ejemplificar las propiedades del entrelazamiento cuántico. Supongamos un gato en una caja metálica con un dispositivo que libera veneno en función del estado de un qubit formado por un átomo. Si el átomo está en un estado predeterminado, el veneno se libera y el gato muere. Como ya hemos visto, un qubit puede estar en una superposición de estados, de modo que cabe preguntarse qué ocurre con el gato. Este experimento pretende representar los efectos del no determinismo cuántico en el mundo macroscópico. La pregunta es: ¿Qué ocurre con el gato antes de que comprobemos si está vivo o muerto? Si el átomo está en superposición, ¿entonces el gato estaría vivo y muerto a la vez? Ya tenemos las herramientas para dar respuesta a esta pregunta.

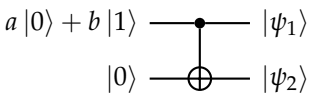
En primer lugar, codificaremos el estado del gato con los vectores $|0\rangle$ y $|1\rangle$ de la base computacional para representar el que esté vivo o muerto, respectivamente, y llamaremos a este sistema G. Por otro lado, utilizaremos la misma base para describir el sistema del átomo, que llamaremos A. Supondremos que el veneno se libera si el estado de A es $|1\rangle$. En ambos sistemas utilizaremos el observable:

$$M = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} = \frac{1}{2} |0\rangle \langle 0| + \frac{1}{2} |1\rangle \langle 1| .$$

¿Cuáles son los estados posibles del sistema? Si inicialmente el estado de A es $a|0\rangle + b|1\rangle$, entonces, los estados de $A \otimes G$ son:

- $|00\rangle$: el veneno no se ha liberado y, por lo tanto, el gato está vivo. Este estado ocurre con probabilidad a^2.
- $|01\rangle$: el gato ha muerto sin que se haya liberado el veneno. Este estado no se da.
- $|10\rangle$: el gato está vivo habiéndose liberado el veneno. Este estado no se da.
- $|11\rangle$: el veneno se ha liberado y por ende el gato ha muerto. Este estado se da con probabilidad b^2.

Por lo tanto, el estado de $A \otimes G$ será la superposición $a|00\rangle + b|11\rangle$. Una forma alternativa de conseguir este resultado es utilizando un circuito con la puerta CNOT de la misma forma que explicamos en la sección 3.5:



No sabemos los estados concretos $|\psi_1\rangle$ y $|\psi_2\rangle$, pero sí que sabemos $|\psi_1\rangle \otimes |\psi_2\rangle = a|00\rangle + b|11\rangle$.

Hemos utilizado una puerta CNOT con el qubit de control en superposición, así que los dos qubits en la salida quedan entrelazados. Esto se puede probar fácilmente usando la definición 3.2. Sin embargo, esto no responde a la pregunta original: ¿El gato está vivo, muerto o ambos a la vez? Para responder a esta pregunta utilizaremos el operador densidad reducido. Nuestro sistema se encuentra en un estado puro: $a|00\rangle + b|11\rangle$, de modo que su matriz densidad queda:

$$\rho = (a|00\rangle + b|11\rangle)(a\langle 00| + b\langle 11|) = a^2|00\rangle\langle 00| + ab|11\rangle\langle 00| + ab|00\rangle\langle 11| + b^2|11\rangle\langle 11|.$$

Ahora, tomamos la traza parcial sobre el estado del átomo, de manera que el estado del gato es:

$$\begin{aligned}\rho_G = tr_1(a^2|00\rangle\langle 00|) + tr_1(ab|11\rangle\langle 00|) + tr_1(ab|00\rangle\langle 11|) + tr_1(b^2|11\rangle\langle 11|) = \\ a^2|0\rangle\langle 0|\langle 0|0\rangle + ab|1\rangle\langle 0|\langle 0|1\rangle + ab|0\rangle\langle 1|\langle 1|0\rangle + b^2|1\rangle\langle 1|\langle 1|1\rangle = \\ a^2|0\rangle\langle 0| + b^2|1\rangle\langle 1| = \begin{bmatrix} a^2 & 0 \\ 0 & b^2 \end{bmatrix}.\end{aligned}$$

Como $tr((\rho_G)^2) = |a|^4 + |b|^4 < 1$ este sistema se encuentra en un estado mezcla, que nos dice, básicamente, que podemos encontrarnos con el gato sin vida con una probabilidad b^2 y vivo con una probabilidad a^2. Es de extrema importancia notar que pese a la creencia popular, el gato no está en superposición, ya que no es una partícula subatómica susceptible a estarlo, sino que se encuentra en un estado mezcla que simplemente no conocemos. Esto pone de manifiesto un matiz importante: las probabilidades que utilizamos como escalar en el operador densidad no tienen el mismo carácter que las amplitudes de los vectores

estado: el no determinismo que representan los primeros vienen de la falta de información del sistema, mientras que en el caso de los vectores estado, el no determinismo es intrínseco al mismo y es producto de las leyes físicas que lo gobiernan.

3.7. Decomposición de Schmidt

La descomposición de Schmidt es otra de las herramientas que se utilizan para estudiar sistemas compuestos, especialmente aquellos en los que el entrelazamiento esté presente. Empezamos con la enunciación del teorema que nos da la herramienta.

Teorema 3.3. *Supongamos $|\psi\rangle$ un estado puro de un sistema compuesto, $s_1 \otimes s_2$ en el que los espacios vectoriales asociados a s_1 y s_2 tienen la misma dimensión. Entonces, existen bases ortonormales $B_{s_1} = \{|i_{s_1}\rangle\}$ y $B_{s_2} = \{|i_{s_2}\rangle\}$, con $i = 1, \ldots, n$ tales que:*

$$|\psi\rangle = \sum_{i=0}^{n} \lambda_i \, |i_{s_1}\rangle \, |i_{s_2}\rangle,$$

donde los coeficientes λ_i son números reales no negativos que satisfacen $\sum_{i=0}^{n} \lambda_i^2 = 1$ denominados coeficientes de Schmidt.

Demostración. Supongamos que los espacios vectoriales asociados a s_1 y a s_2 tienen la misma dimensión. Cojamos dos bases ortonormales cualesquiera B_{s_i} para s_i donde $i = 1, 2$. Dado que el producto de Kronecker de bases es una base, el estado $|\psi\rangle$ puede escribirse entonces como:

$$|\psi\rangle = \sum_{i=0}^{n} \sum_{j=0}^{n} m_{ij} \, |i_{s_1}\rangle \, |j_{s_2}\rangle,$$

siendo m_{ij} una matriz unitaria. Por la proposición 1.5 existen matrices u_{ij},d_{ij} y v_{ij} tales que $m_{ij} = u_{ij} d_{ij} v_{ij}$, siendo d_{ij} diagonal con entradas no negativas. Si expresamos la multiplicación de estas matrices elemento a elemento y sustituimos en la expresión anterior obtenemos:

$$|\psi\rangle = \sum_{i,j,k=0}^{n} u_{jk} d_{kk} v_{ki} \, |i_{s_1}\rangle \, |j_{s_2}\rangle$$

Si ahora reescribimos: $|k_{s_2}\rangle = \sum_{j=0}^{n} u_{jk} \, |i_{s_2}\rangle$, $|k_{s_1}\rangle = \sum_{j=0}^{n} v_{ki} \, |i_{s_1}\rangle$ y $d_{kk} = \lambda_k$. Entonces nos queda la expresión equivalente a lo que queríamos probar:

$$|\psi\rangle = \sum_{k=0}^{n} \lambda_k \, |k_{s_1}\rangle \, |k_{s_2}\rangle.$$

□

Una consecuencia muy útil de este teorema es que los operadores de densidad reducidos de un sistema compuesto por dos qubits tienen los mismos autovalores. Vamos a comprobarlo. Sea $|\psi\rangle$ el estado puro de un sistema compuesto, $s_1 s_2$. Tomando como bases B_{s_1} y B_{s_2},

realizamos la traza parcial a cada uno de los sistemas, quedando:

$$\rho_{s_1} = tr_{s_2}(|\psi\rangle \langle\psi|) = \sum_{i=0}^{n} \lambda_i |i_{s_1}\rangle \langle i_{s_1}|,$$
$$\rho_{s_2} = tr_{s_1}(|\psi\rangle \langle\psi|) = \sum_{i=0}^{n} \lambda_i |i_{s_2}\rangle \langle i_{s_2}|.$$

1 Podemos ver claramente que λ_i son autovalores comunes de las respectivas matrices de densidad reducidas de los subsistemas s_1 y s_2.

Esta consecuencia es importante, puesto que muchas propiedades de un subsistema cuántico quedan determinadas por los autovalores de la matriz densidad reducida del mismo. Un ejemplo de esto es la traza. Si los autovalores son los mismos, la traza es la misma, de modo que propiedades como la esperanza son iguales.

4 Computación cuántica

Hasta ahora nos hemos centrado en explicar los fenómenos del mundo subatómico y las reglas que gobiernan los procesos que involucran partículas extremadamente pequeñas. En este capítulo explicaremos como podemos utilizar los fenómenos cuánticos para la implementación física de nuevos algoritmos.

Los sistemas cuánticos en los que nos centraremos son los qubits y la composición de los mismos. En primer lugar, presentaremos las puertas lógicas cuánticas, que no son más que una representación de las distintas transformaciones unitarias que se pueden realizar en un qubit. Después, pasaremos a los circuitos cuánticos: conjuntos de puertas que realizan operaciones más complejas. Por último, pasaremos a los algoritmos cuánticos, que usan estos circuitos para obtener resultados prácticos.

De la misma forma que en el capítulo anterior, la mayor parte de los conceptos que se discuten en este capítulo provienen de la fuente [NC10].

4.1. Introducción a los circuitos cuánticos

4.1.1. Puertas lógicas cuánticas

Como ya hemos mencionado, una puerta lógica cuántica es una transformación unitaria. Además, cualquier matriz unitaria puede ser una puerta cuántica. Sin embargo, nos centraremos solo en las más importantes, como la puerta de Hadamard (1.1), la puerta CNOT (2.4) o las puertas de Pauli (3.2), entre otras.

Ejemplo 4.1. Tomamos la puerta de Hadamard. Aplicar esta puerta a un solo qubit se traduce en multiplicar la matriz asociada a la puerta por el vector estado del qubit. Supongamos un vector estado en la base computacional $|\psi\rangle = |0\rangle$. Entonces el resultado de aplicar la transformación es:

$$H|\psi\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\end{bmatrix} + \frac{1}{\sqrt{2}}\begin{bmatrix}0\\1\end{bmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle.$$

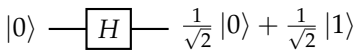


Figura 4.1: Circuito formado por una única puerta Hadamard que crea superposición a partir de un estado a partir de un estado de la base computacional.

Ejemplo 4.2. Tomamos ahora la puerta CNOT. Esta puerta se aplica a dos qubits, tomemos q_1 y q_2, con sus respectivos estados $|\psi_1\rangle = a_1|0\rangle + b_1|1\rangle$ y $|\psi_2\rangle = a_2|0\rangle + b_2|1\rangle$. Recordemos

que la operación de la puerta CNOT es $f(x_1, x_2) = (x_1, x_1 \oplus x_2)$. La matriz asociada a esta puerta es:

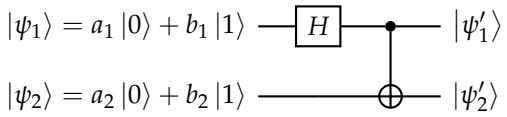
$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Esta matriz se aplica al sistema completo formado por los dos qubits q_1 y q_2, de manera que para calcular el resultado, tendremos que tomar $|\psi_1\rangle \otimes |\psi_2\rangle = a_1a_2 |00\rangle + a_1b_2 |01\rangle + b_1a_2 |10\rangle + b_1b_2 |11\rangle$. De modo que aplicar la matriz de la puerta al vector $|\psi_1\rangle \otimes |\psi_2\rangle$ queda:

$$\text{CNOT}(|\psi_1\rangle \otimes |\psi_2\rangle) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_1a_2 \\ a_1b_2 \\ b_1a_2 \\ b_1b_2 \end{bmatrix} = \begin{bmatrix} a_1a_2 \\ a_1b_2 \\ b_1b_2 \\ b_1a_2 \end{bmatrix}.$$

De esta manera, tal y como esperábamos, la puerta transforma $a_1a_2 |00\rangle + a_1b_2 |01\rangle + b_1a_2 |10\rangle + b_1b_2 |11\rangle$ en $a_1a_2 |00\rangle + a_1b_2 |01\rangle + b_1a_2 |11\rangle + b_1b_2 |10\rangle$. La puerta CNOT es un ejemplo de las $4! = 24$ puertas lógicas reversibles a dos bits que permutan los vectores de la base computacional. Explicaremos detenidamente este hecho más adelante.

Ejemplo 4.3. Ya tenemos las herramientas necesarias para formalizar el circuito 3.5 que introdujimos en la sección 3.5. Supongamos que tenemos el siguiente circuito:



Vamos a calcular la salida de este circuito. En primer lugar tenemos que lidiar con la primera operación, que es un una puerta Hadamard en el primer qubit y nada en la segunda. El no hacer nada es lo mismo que multiplicar el estado por la matriz identidad 2×2. Si queremos aplicar dos operaciones distintas, entonces la operación sobre el sistema completo será el producto de Kronecker de las mismas, en este caso $(H \otimes I)$. De este modo, vemos que

el estado del sistema tras aplicar esta primera operación será:

$$(H \otimes I_{2\times 2})(|\psi_1\rangle \otimes |\psi_2\rangle) = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \otimes \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} (a_1a_2\,|00\rangle + a_1b_2\,|01\rangle + b_1a_2\,|10\rangle + b_1b_2\,|11\rangle)$$

$$= \frac{1}{\sqrt{2}}\begin{bmatrix} 1\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} & 1\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \\ 1\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} & -1\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\end{bmatrix}\begin{bmatrix}a_1\\ b_1\\ a_2\\ b_2\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & -1 & 1 & -1\\ 1 & 1 & -1 & -1\\ 1 & -1 & -1 & 1\end{bmatrix}\begin{bmatrix}a_1\\ b_1\\ a_2\\ b_2\end{bmatrix} =$$

$$\frac{1}{\sqrt{2}}\begin{bmatrix}a_1+b_1+a_2+b_2\\ a_1-b_1+a_2-b_2\\ a_1+b_1-a_2-b_2\\ a_1-b_1-a_2+b_2\end{bmatrix} =$$

$$\frac{a_1+b_1+a_2+b_2}{\sqrt{2}}\,|00\rangle + \frac{a_1-b_1+a_2-b_2}{\sqrt{2}}\,|01\rangle + \frac{a_1+b_1-a_2-b_2}{\sqrt{2}}\,|10\rangle + \frac{a_1-b_1-a_2+b_2}{\sqrt{2}}\,|11\rangle\,.$$

A continuación aplicamos la puerta CNOT:

$$\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix}a_1+b_1+a_2+b_2\\ a_1-b_1+a_2-b_2\\ a_1+b_1-a_2-b_2\\ a_1-b_1-a_2+b_2\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}a_1+b_1+a_2+b_2\\ a_1-b_1+a_2-b_2\\ a_1-b_1-a_2+b_2\\ a_1+b_1-a_2-b_2\end{bmatrix}.$$

Ahora, si tomamos todas las combinaciones posibles para $a_i = 0$, $b_j = 0$ para $i, j = 1, 2$, obtenemos exactamente los mismos resultados que obtuvimos la primera vez que vimos este circuito con el objetivo de estudiar el entrelazamiento cuántico.

Ya explicamos en el capítulo 3 que los vectores estado puro de un qubit pertenecen a la *esfera de Bloch*. Las transformaciones unitarias de un vector mantienen su módulo, por lo que el resultado será otro vector unitario también en la esfera de Bloch y por ende, la operación puede verse como una *rotación*. Para definir las rotaciones en la esfera de esta esfera, utilizaremos unas nuevas puertas denominadas *operadores de rotación* que se definen a partir de las puertas de Pauli (3.2) y las funciones operador (1.12). Dado $\theta \in \mathbb{R}$:

$$R_x(\theta) := e^{-i\theta\frac{X}{2}} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}X = \begin{bmatrix}\cos\frac{\theta}{2} & -i\sin\frac{\theta}{2}\\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2}\end{bmatrix},$$

$$R_y(\theta) := e^{-i\theta\frac{Y}{2}} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}Y = \begin{bmatrix}\cos\frac{\theta}{2} & -\sin\frac{\theta}{2}\\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2}\end{bmatrix},$$

$$R_z(\theta) := e^{-i\theta\frac{Z}{2}} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}Z = \begin{bmatrix}e^{-i\frac{\theta}{2}} & 0\\ 0 & e^{i\frac{\theta}{2}}\end{bmatrix}.$$

Estas matrices representan las rotaciones de ángulo θ respecto de los ejes X, Y y Z respectivamente. Los operadores de rotación cumplen una particularidad especial: todas las transformaciones unitarias pueden describirse a partir de ellas.

Teorema 4.1 (Descomposición Z-Y). *Para toda matriz unitaria compleja U de tamaño 2 existen números reales α, β, γ y δ tales que:*

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

Demostración. Como las columnas de una matriz unitaria forman vectores ortonormales, entonces la matriz U puede escribirse como:

$$U = \begin{bmatrix} e^{i\lambda_1} \cos\varphi & -e^{i\lambda_2} \sin\varphi \\ e^{i\lambda_3} \sin\varphi & e^{i\lambda_4} \cos\varphi \end{bmatrix}.$$

Siendo $\lambda_i, \varphi \in \mathbb{R}$ tales que $\lambda_1 + \lambda_4 = \lambda_2 + \lambda_3$. Ahora, reescribiremos dichos números reales como:

$$\lambda_1 = \alpha - \frac{\beta}{2} - \frac{\delta}{2}, \lambda_2 = \alpha - \frac{\beta}{2} + \frac{\delta}{2},$$
$$\lambda_3 = \alpha + \frac{\beta}{2} - \frac{\delta}{2}, \lambda_4 = \alpha + \frac{\beta}{2} + \frac{\delta}{2},$$
$$\theta = \frac{\gamma}{2}.$$

Por lo que reescribimos U con las identidades anteriores.

$$U = \begin{bmatrix} e^{i(\alpha - \frac{\beta}{2} - \frac{\delta}{2})} \cos\frac{\gamma}{2} & -e^{i(\alpha - \frac{\beta}{2} + \frac{\delta}{2})} \sin\frac{\gamma}{2} \\ e^{i(\alpha + \frac{\beta}{2} - \frac{\delta}{2})} \sin\frac{\gamma}{2} & e^{i(\alpha + \frac{\beta}{2} + \frac{\delta}{2})} \cos\frac{\gamma}{2} \end{bmatrix}.$$

A continuación, descomponemos U:

$$U = e^{i\alpha} \begin{bmatrix} e^{i(-\frac{\beta}{2} - \frac{\delta}{2})} \cos\frac{\gamma}{2} & -e^{i(-\frac{\beta}{2} + \frac{\delta}{2})} \sin\frac{\gamma}{2} \\ e^{i(\frac{\beta}{2} - \frac{\delta}{2})} \sin\frac{\gamma}{2} & e^{i(\frac{\beta}{2} + \frac{\delta}{2})} \cos\frac{\gamma}{2} \end{bmatrix} = \begin{bmatrix} e^{-i\frac{\beta}{2}} & 0 \\ 0 & e^{\frac{\beta}{2}} \end{bmatrix} \begin{bmatrix} \cos\frac{\gamma}{2} & -\sin\frac{\gamma}{2} \\ \sin\frac{\gamma}{2} & \cos\frac{\gamma}{2} \end{bmatrix} \begin{bmatrix} e^{-i\frac{\delta}{2}} & 0 \\ 0 & e^{\frac{\delta}{2}} \end{bmatrix}$$

Pero podemos observar que claramente esta es la identidad:

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

□

En general, no utilizaremos este teorema, sino una consecuencia más general del mismo.

Corolario 4.1. *Dada una puerta lógica cuántica U que actúa sobre un qubit, existen operadores unitarios A, B,C y $\alpha \in \mathbb{R}$ tales que $ABC = I$ y $U = e^{i\alpha} AXBXC$.*

Demostración. En las condiciones del teorema 4.1, definimos los operadores:

$$A = R_z(\beta) R_y\left(\frac{\gamma}{2}\right),$$
$$B = R_y\left(-\frac{\gamma}{2}\right) R_z\left(-\frac{\delta + \beta}{2}\right),$$
$$C = R_z\left(\frac{\delta - \beta}{2}\right)$$

$$ABC = R_z(\beta)\, R_y\left(\frac{\gamma}{2}\right) R_y\left(-\frac{\gamma}{2}\right) R_z\left(-\frac{\delta+\beta}{2}\right) R_z\left(\frac{\delta-\beta}{2}\right) = I.$$

Por otro lado, tenemos:

$$XR_y\left(\frac{\gamma}{2}\right)X = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}\cos\frac{\gamma}{2} & -\sin\frac{\gamma}{2}\\ \sin\frac{\gamma}{2} & \cos\frac{\gamma}{2}\end{bmatrix}\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} = \begin{bmatrix}\cos\frac{\gamma}{2} & \sin\frac{\gamma}{2}\\ -\sin\frac{\gamma}{2} & \cos\frac{\gamma}{2}\end{bmatrix} = R_y\left(\frac{\gamma}{2}\right)^{-1} = R_y\left(-\frac{\gamma}{2}\right)$$

$$XR_z\left(-\frac{\delta+\beta}{2}\right)X = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}e^{i\frac{\delta+\beta}{2}} & 0\\ 0 & e^{-i\frac{\delta+\beta}{2}}\end{bmatrix}\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} = \begin{bmatrix}e^{-i\frac{\delta+\beta}{2}} & 0\\ 0 & e^{i\frac{\delta+\beta}{2}}\end{bmatrix} = R_z\left(-\frac{\delta+\beta}{2}\right)^{-1} = R_y\left(\frac{\delta+\beta}{2}\right).$$

Ahora, teniendo en cuenta que $X^2 = I$, tenemos que:

$$\begin{aligned} XBX &= XR_y\left(-\frac{\gamma}{2}\right) R_z\left(-\frac{\delta+\beta}{2}\right) X \\ &= XR_y\left(-\frac{\gamma}{2}\right) XXR_z\left(-\frac{\delta+\beta}{2}\right) X \\ &= R_y\left(\frac{\gamma}{2}\right) R_z\left(\frac{\delta+\beta}{2}\right). \end{aligned}$$

Finalmente, calculamos $AXBXC$.

$$AXBXC = R_z(\beta)\, R_y\left(\frac{\gamma}{2}\right) R_y\left(\frac{\gamma}{2}\right) R_z\left(\frac{\delta+\beta}{2}\right) R_z\left(\frac{\delta-\beta}{2}\right) = R_z(\beta)R_y(\gamma)R_z(\delta).$$

Por lo tanto, si multiplicamos por $e^{i\alpha}$ con α proporcionado por el teorema 4.1, obtenemos $U = e^{i\alpha}R_z(\beta)R_y(\gamma)R_z(\delta) = e^{i\alpha}AXBXC$, que es el resultado buscado. □

Por otra parte, una formulación parecida de esta descomposición nos servirá de gran utilidad más adelante, cuando tratemos la universalidad de las puertas cuánticas.

Proposición 4.1. *Dados vectores $\vec{m}$ y $\vec{n}$ en $\mathbb{R}^3$ no paralelos, para toda puerta lógica cuántica existen $\alpha, \beta, \gamma, \delta \in \mathbb{R}$ tales que:*

$$U = e^{i\alpha}R_{\vec{n}}(\beta)R_{\vec{m}}(\gamma)R_{\vec{n}},(\delta).$$

donde $R_{\vec{n}}(\beta)$ es la rotación de β grados respecto al eje formado por el vector $\vec{n}$.

4.1.2. Simulación de circuitos convencionales

Es importante notar que si no utilizamos ningún efecto cuántico, las puertas lógicas cuánticas pueden funcionar como puertas lógicas convencionales. La única restricción es que las transformaciones unitarias permitidas en este caso son permutaciones de las filas de la matriz identidad ¿A qué se debe esto? Si codificamos los estados 0 y 1 del bit convencional como los vectores $|0\rangle$ y $|1\rangle$ de la base computacional, entonces aplicarle una permutación a cualquiera

de estos nos lleva el uno en el otro. Es decir:

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

Esta operación no es más que la negación booleana. Pero este es el caso en el que tenemos un solo qubit, vamos a probar que esto ocurre también cuando tenemos más de uno. La base computacional de n qubits tiene la forma $B = \{|10\ldots0\rangle, \ldots, |0\ldots1\ldots0\rangle, \ldots, |0\ldots1\rangle\}$. Si multiplicamos estos vectores por la matriz resultante de permutar las filas i_1 e i_2 , provoca esa misma permutación en el vector resultado. De esta manera, multiplicar un vector de la base por la matriz permutada nos lleva a otro vector de la base.

Por otra parte, las columnas de una matriz unitaria general forman una base ortonormal que no tiene por qué ser la computacional y por ende, genera superposición de estados de la misma.

De esta manera, probamos que si identificamos los vectores $|a_1, \ldots, a_n\rangle$ con un conjunto de bits $a_1 \ldots a_n$, cualquier función booleana multivaluada reversible puede verse simulada por una transformación unitaria descrita por una permutación de las filas de la matriz identidad. Esto es de extrema importancia, porque nos permite describir cómo un circuito cuántico puede simular un circuito reversible clásico.

4.1.3. Operaciones condicionales

Otra herramienta que resulta de mucha utilidad son las operaciones condicionales. Ya hemos visto la operación CNOT, que es un caso particular de estas. Ya hemos visto que la matriz de esta operación es:

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

Sin embargo, ¿De dónde sale esta matriz? Daremos una construcción intuitiva de la misma. Vamos a tomar $P_0 = |0\rangle\langle 0|$ y $P_1 = |1\rangle\langle 1|$. Estas operaciones son las proyecciones a los subespacios generados por los vectores $|0\rangle$ y $|1\rangle$ respectivamente. Como estos vectores son ortonormales, claramente $P_i |j\rangle = \vec{0}$ si $i \neq j$ y $P_i |j\rangle = |j\rangle$ en caso contrario. Queremos aplicar la operación negación, definida por el operador X, si el qubit de control está en el estado $|1\rangle$ y no hacer nada si está en el estado $|0\rangle$ por lo que "asociamos" estos casos haciendo el cálculo $P_0 \otimes I$ y $P_1 \otimes X$. Este cálculo provoca que el primer qubit se anule cuando no está en el caso deseado, generando que esa componente de la suma no se aplique. Calculemos explícitamente el resultado para aclarar este concepto. Si tenemos los qubits de control y objetivo $|c\rangle$ y $|d\rangle$, entonces el estado del sistema tras la operacion es:

$$|\psi\rangle = (P_0 \otimes I + P_1 \otimes X)(|c\rangle \otimes |d\rangle) = P_0 |c\rangle \otimes I |d\rangle + P_1 |c\rangle \otimes X |d\rangle . \tag{4.1}$$

Es fácil ver que cuando $|c\rangle = |0\rangle$ entonces, $|\psi\rangle = |c\rangle \otimes |d\rangle = |c\rangle \otimes |d\rangle$. Por otro lado, si $|c\rangle = |1\rangle$, entonces $|\psi\rangle = |c\rangle \otimes X |d\rangle = |c\rangle \otimes X |d\rangle$, que es justo la operación que realizamos al aplicar la puerta CNOT. Además, si calculamos la matriz resultante:

$$P_0 \otimes I + P_1 \otimes X = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

Nos da justo la matriz asociada a esta puerta lógica. Por otro lado, esta operación condicional puede extenderse a cualquier operador.

La puerta CNOT puede verse como una función que sale y llega al espacio vectorial de estados del sistema compuesto, siguiendo la regla: $|c\rangle |t\rangle \rightarrow |c\rangle |c \oplus t\rangle$. Esta regla puede generalizarse para una transformación unitaria, U, cualquiera como una función que se escribiría como $|c\rangle |t\rangle \rightarrow |c\rangle U^c |t\rangle$, siendo $U^1 = U$ y $U^0 = I$. Su representación gráfica en un circuito sería:

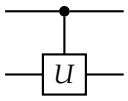


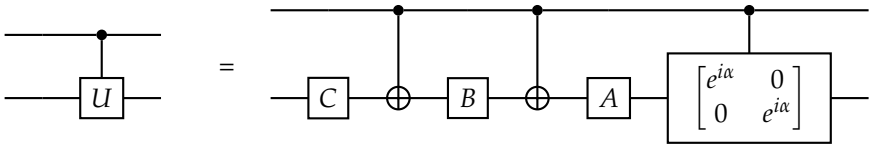
Figura 4.2: Circuito simple con operación condicional U

A continuación daremos una posible implementación para esta operación condicional utilizando únicamente puertas lógicas de un solo qubit y la puerta CNOT gracias al corola-

rio 4.1. En primer lugar, utilizamos el cálculo expuesto en (4.1) para construir el operador condicional M_α, que tiene como matriz:

$$M_\alpha = P_0 \otimes I + P_1 \otimes \begin{bmatrix} e^{i\alpha} & 0 \\ 0 & e^{i\alpha} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{i\alpha} & 0 \\ 0 & 0 & 0 & e^{i\alpha}. \end{bmatrix}$$

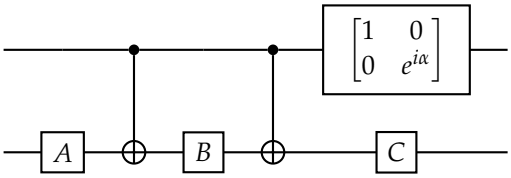
De este modo, dados los operadores A, B, C y el número real α proporcionados por el corolario 4.1 para escribir cualquier operación unitaria U, obtenemos el circuito para U controlada:



Además, la matriz del operador M_α puede escribirse como:

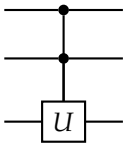
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{i\alpha} & 0 \\ 0 & 0 & 0 & e^{i\alpha} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{bmatrix} \otimes I = \tilde{M}_\alpha \otimes I.$$

De este modo, podemos reescribir el circuito anterior como:

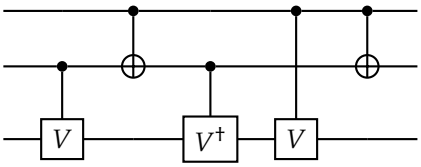


Cuando el primer qubit está en el estado $|0\rangle$, entonces $\tilde{M}_\alpha |0\rangle = |0\rangle$ en el primer bit, mientras que en el segundo se aplica la transformacion ABC, que sabemos que da como resultado la identidad. Por otro lado, si el primer qubit está en el estado $|1\rangle$, entonces está claro que en el segundo se aplica la transformación $AXBXCe^{i\alpha} = U$, por lo que este circuito es equivalente al de la figura 4.1.3.

Por otro lado, también podemos definir operaciones condicionales a dos qubits. Esto se representaría gráficamente con el circuito:



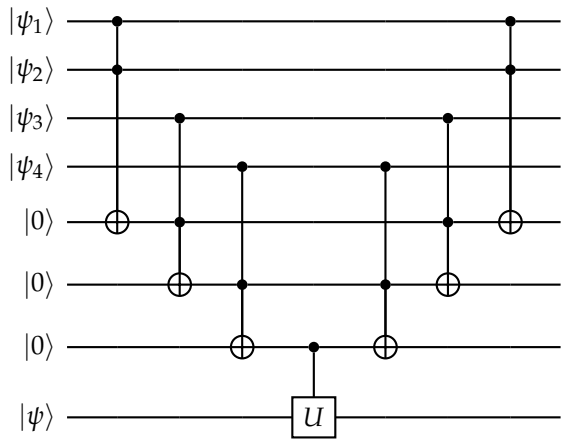
La manera en la que implementamos este circuito es buscando un operador unitario V tal que $V^2 = U$ y construyendo en siguiente circuito:



Si revisamos la casuística de este circuito, vemos que se cumple la condición de que V solo se aplica cuando los dos primeros qubits están en el estado $|1\rangle$. Un caso específico bastate usual es la implementación de la puerta U = Toffoli o CCNOT, que se consigue usando $V = \frac{1-i}{2}(I + iX)$. También podemos escribir la puerta Toffoli explícitamente con proyectores de la misma manera que lo hemos hecho con la puerta CNOT:

$$T = (P_{00} \otimes I) + (P_{10} \otimes I) + (P_{01} \otimes I) + (P_{11} \otimes X) =$$
$$(P_0 \otimes P_0 \otimes I) + (P_1 \otimes P_0 \otimes I) + (P_0 \otimes P_1 \otimes I) + (P_1 \otimes P_1 \otimes X) =$$
$$\begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix} \otimes I + \begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&0\end{bmatrix} \otimes I + \begin{bmatrix}0&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix} \otimes I + \begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&1\end{bmatrix} \otimes X =$$
$$\begin{bmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&1\\0&0&0&0&0&0&1&0\end{bmatrix}$$

Por último, también es posible hacer depender una puerta de n qubits. Esto puede hacerse mediante una estructura simple pero que sin embargo requiere del uso de $n-1$ qubits prefijados a $|0\rangle$ que se usan únicamente para cálculo. Esta estructura se consigue mediante puertas Toffoli interdependientes. Un ejemplo para $n = 4$ es:

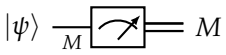


De nuevo, es fácil comprobar la casuística del circuito para ver que solo se aplica la puerta condicional U cuando $|\psi_i\rangle = |1\rangle$ para $i = 1, 2, 3, 4$. Esta construcción puede extenderse a cualquier número de qubits de control.

4.1.4. Medida

La medida de un qubit es otra parte fundamental de los circuitos cuánticos que los diferencian de los circuitos convencionales. En general, siempre realizaremos mediciones sobre la base computacional, puesto que realizarlas en cualquier otra base es lo mismo que aplicar una operación unitaria que represente un cambio de base antes de medir.

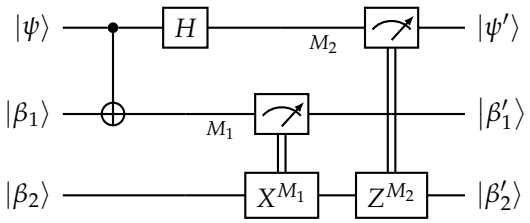
Como ya hemos visto en el capítulo 4, sabemos que el medir un qubit en la base computacional arroja como resultado un valor que se corresponde con un bit clásico. Mientras que los cables de los circuitos que transmiten información cuántica se representan mediante un solo cable, aquellos que llevan información convencional se dibujan con un cable doble. La medición se representa gráficamente de la siguiente manera:



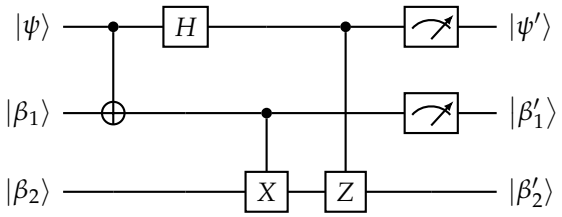
La variable M hace referencia al valor producido por la medida. La medición en un circuito cuántico puede hacerse de varias maneras. Sin embargo, existen 2 principios que utilizamos con el fin de simplificar el proceso.

- **Principio de medida diferida:** la medida siempre puede posponerse hasta el final del circuito. Si los resultados se utilizan para alguna puerta lógica intermedia, esta puede cambiarse por una puerta condicional.
- **Principio de medida implícita:** cualquier cable de un circuito en el que pueda quedar un estado indeterminado, puede suponerse como medido, puesto que el hacerlo no supondría un cambio en la distribución estadística de las medidas previamente realizadas.

Ejemplo 4.4. Veamos ahora un ejemplo que ponga de manifiesto estos principios. Tomemos el siguiente circuito:



Este circuito presenta varios conceptos nuevos. En primer lugar, observamos que el valor de las mediciones, M_i, se están usando para condicionar otras puertas. Esto es claro ya que si, por ejemplo $M_1 = 0$, entonces $X^0 = I$, mientras que si $M_1 = 1$, $X^1 = X$. Por otro lado, vemos que salen circuitos cuánticos de las mediciones, al contrario que en el circuito 4.1.4, más simple. Esto es debido a que como vimos en el capítulo 4, la medición modifica el estado del qubit, convirtiendo el estado previo en uno de la base computacional. A continuación, utilizamos el primer principio para realizar las mediciones al final, consiguiendo el circuito:



La equivalencia de estos circuitos viene del hecho de que el cable que tiene como entrada el qubit $|\beta_2\rangle$ pasa por dos puertas condicionales. Por tanto, el estado de este qubit al final de este cable viene dado por los estados de los demás qubits. Al realizar la medición al final de los cables de estos, su estado queda determinado, por lo que el estado del tercer qubit también queda decidido.

Por otro lado, el segundo principio nos dice que podemos suponer que el tercer qubit queda medido al final del cable. Al haber medido los demás qubits, estos quedan determinados y el hecho de medir también en el tercero no cambia en nada estos resultados.

4.2. Universalidad

En el capítulo 2 ya vimos como un subconjunto de puertas lógicas clásicas podían generar, mediante composiciones y proyecciones, cualquier otra puerta. En esta sección presentaremos el concepto equivalente para puertas lógicas cuánticas. Sin embargo, esto plantea dificultades. En la versión convencional, el número de puertas lógicas reversibles de tamaño n es finito, mientras que en el caso cuántico, el número de operadores unitarios de n qubits es el número de matrices unitarias de tamaño 2^n , es decir, *infinitas incontables*. Es por este motivo que es imposible encontrar un conjunto finito de puertas que las puedan generar de forma exacta,

pero sí que existen conjuntos finitos que las puedan aproximar.

Definición 4.1. Una matriz de tamaño d se dice *de dos niveles* si, al multiplicar por un vector de tamaño d, solo actúa de forma no trivial en dos o menos de las componentes del vector.

Las matrices unitarias de dos niveles pueden escribirse como matrices identidad en las que se sustituye una submatriz 2×2 de la diagonal por otra transformación unitaria.

Ejemplo 4.5. La multiplicación de matriz y vector:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix},$$

solo afecta a las componentes a_2 y a_3.

Estas matrices son de extrema importancia, puesto que nos permiten construir cualquier otra matriz unitaria.

Teorema 4.2. *Una matriz unitaria de tamaño d puede expresarse como producto de matrices de dos niveles.*

Demostración. Empecemos con el caso más simple, en el que $d = 3$, por lo que tenemos la matriz:

$$\begin{bmatrix} a & d & g \\ b & e & h \\ c & f & j \end{bmatrix}.$$

La idea principal de la demostración será encontrar matrices unitarias de dos niveles U_i con $i = 1, 2, 3$ tales que $U_1 U_2 U_3 U = I$, de modo que $U_1^\dagger U_2^\dagger U_3^\dagger = U$. Utilizamos un proceso constructivo:

$$\text{si } b = 0 \text{ entonces } U_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ ,si } b \neq 0 \text{ entonces } U_1 = \begin{bmatrix} \frac{a^*}{\sqrt{|a|^2+|b|^2}} & \frac{b^*}{\sqrt{|a|^2+|b|^2}} & 0 \\ \frac{b}{\sqrt{|a|^2+|b|^2}} & \frac{-a}{\sqrt{|a|^2+|b|^2}} & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

En cualquier caso, U_1 es una matriz de dos niveles. Al multiplicar $U_1 U$ obtenemos:

$$\begin{bmatrix} a' & d' & g' \\ 0 & e' & h' \\ c' & f' & j' \end{bmatrix}.$$

$$\text{Si } c' = 0 \text{ entonces } U_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ y si } c' \neq 0 \text{ entonces } U_2 = \begin{bmatrix} \frac{a'^*}{\sqrt{|a'|^2+|c'|^2}} & 0 & \frac{c'^*}{\sqrt{|a'|^2+|c'|^2}} \\ 0 & 1 & 0 \\ \frac{c'}{\sqrt{|a'|^2+|c'|^2}} & 0 & \frac{-a'}{\sqrt{|a'|^2+|c'|^2}} \end{bmatrix}.$$

En cualquiera de los casos, al multiplicar obtenemos:

$$U_2U_1U = \begin{bmatrix} 1 & d'' & g'' \\ 0 & e'' & h'' \\ 0 & f'' & j'' \end{bmatrix}.$$

Pero como esta matriz es unitaria, entonces debe darse que $d'' = g'' = 0$. Así que la matriz queda:

$$U_2U_1U = \begin{bmatrix} 1 & 0 & 0 \\ 0 & e'' & h'' \\ 0 & f'' & j'' \end{bmatrix}.$$

A partir de aquí, sabiendo que la submatriz formada por la esquina inferior derecha es unitaria, podemos definir U_3 como:

$$U_2U_1U = \begin{bmatrix} 1 & 0 & 0 \\ 0 & e''^* & f''^* \\ 0 & h''^* & j''^* \end{bmatrix}.$$

De modo que hemos encontrado las matrices que buscábamos. Ahora, para $d > 3$, podemos usar el mismo procedimiento en el que vamos multiplicando por matrices de dos niveles para quedarnos con una matriz tal que la primera fila y columna solo tenga un 1. A continuación, aplicamos el mismo proceso en la submatriz formada por las filas y columnas de 2 a d, hasta que quede una única submatriz 2×2, en cuyo caso multiplicamos por una matriz equivalente a la U_3 del caso anterior. □

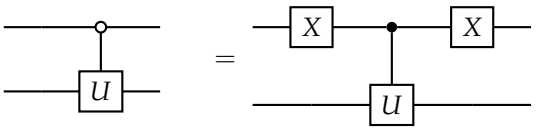
Antes de proceder con el siguiente teorema, es necesario definir un concepto que utilizaremos en la siguiente demostración.

Definición 4.2. Dadas dos secuencias binarias q_1 y q_2 de igual tamaño, definimos el *código de Gray* como un conjunto de secuencias $g_1, \ldots, g_m$ tales que $g_1 = q_1$, $g_m = q_2$ y las secuencias g_i y g_{i+1} se diferencian en un único bit.

Ejemplo 4.6. Un posible código de Gray entre 0001 y 1100 es:

g_1	0	0	0	1
g_2	0	0	0	0
g_3	0	1	0	0
g_4	1	1	0	0

Por otra parte, también es importante mencionar que una puerta controlada tambíen puede estar condicionada a un qubit de control puesto a $|0\rangle$, que se representa gráficamente como:



Teorema 4.3. *Cualquier matriz de dos niveles puede construirse de puertas lógicas cuánticas de un solo qubit y puertas* CNOT.

Demostración. Supongamos una matriz de dos niveles de tamaño n que solo actúa sobre dos vectores de la base computacional, $|s\rangle$ y $|t\rangle$ con las representaciones binarias $s_1, \ldots, s_n$ y $t_1, \ldots, t_n$. Además, sea $\{g_m\}$ un código de Gray de dichas representaciones. También consideramos U como la submatriz no trivial de la matriz de dos niveles, que es una operación sobre un solo qubit. A partir de $\{g_m\}$, construimos un circuito usando el siguiente procedimiento. Para cada i, con $i = 1, \ldots, m-1$, añadimos una operación X controlada en el qubit que en el que g_i se diferencia con g_{i+1}, en el que los qubits de control son todos los demás, que deberían ser idénticos. Para los qubits que sean $|0\rangle$, invertimos el bit de control antes y después. El resultado es que el qubit objetivo solo cambia cuando se da la secuencia exacta que buscamos. Una vez hayamos realizado esta acción hasta $m-1$, la composición de estas puertas realiza la siguiente operación:

$$|g_1\rangle \to |g_{m-1}\rangle\,,\ |g_2\rangle \to |g_1\rangle\,,\ |g_3\rangle \to |g_2\rangle\,,\ \ldots,|g_{m-1}\rangle \to |g_{m-2}\rangle\,.$$

A continuación, realizamos la operación U al bit que diferencia a g_{m-1} y a g_m condicionada al resto de los bits, de manera que solo se aplique si la secuencia es g_{m-1} o g_m. Después, aplicamos todas las puertas condicionales de nuevo, a la inversa.

¿Por qué este procedimiento funciona? Como hemos visto, las puertas condicionales impiden que cualquier secuencia que no se encuentre en el código de Gray se vean afectadas. Y aquellas que si se encuentren, pero no sean $|g_1\rangle$ o $|g_m\rangle$, ven desechos sus cambios después de aplicar las operaciones condicionales a la inversa sin aplicar U. Por el contrario, si las secuencia de entrada es $|g_1\rangle$, entonces las puertas condicionales la convertirán en $|g_{m-1}\rangle$ se aplicará la operación U y se desharán dichos cambios. Esto se puede hacer porque el qubit en el que se aplica la operación no cambia. Por otro lado, si la entrada es $|g_m\rangle$, entonces no se aplica ninguna operación condicional y únicamente se realiza la operación U sobre el bit correspondiente.

Por último, solo cabe destacar que todas las operaciones que utilizamos en la construcción anteriormente explicada puede construirse con puertas sobre un solo qubit y puertas CNOT, como ya hemos visto en secciones anteriores. □

Ya tenemos que cualquier operador unitario de tamaño d puede expresarse *exactamente* como combinación de puertas lógicas sobre un solo qubit y las puerta CNOT. Ahora veremos como podemos *aproximar* cualquier puerta mediante el subconjunto de puertas lógicas:

$$\begin{aligned}
\text{Hadamard} &= \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} = H, \\
\frac{\pi}{8} &= \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{bmatrix} = T, \\
\text{fase} &= \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} = F, \\
\text{y controlled-not} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \text{CNOT}.
\end{aligned}$$

Para poder llevar a cabo esta prueba, necesitamos definir primero el concepto de proximidad en este contexto.

Definición 4.3. Dados operadores unitarios U y V , se define el *error* entre U y V como:

$$E(U,V) = \max_{\||\psi\rangle\|=1} \|(U-V)\,|\psi\rangle\|.$$

Esta función está bien definida puesto que fijados U y V, la función $|\psi\rangle \to \|(U-V)\,|\psi\rangle\|$ es continua y está definida sobre un conjunto compacto, por lo que alcanza su máximo.

Una propiedad interesante es que, como consecuencia de la propiedad triangular, si tenemos operadores unitarios $U_1,\ldots,U_m,V_1,\ldots,V_m$, entonces:

$$E(U_1U_2\ldots U_{m-1}U_m, V_1V_2\ldots V_{m-1}V_m) < \sum_{j=1}^{m} E(U_j,V_j).$$

Teorema 4.4. *$\forall\epsilon > 0$, para todo operador U, existe otro operador R formado por las puertas H,T,F y* CNOT *tales que $E(U,R) < \epsilon$.*

Demostración. La matriz T es una rotación de $\frac{\pi}{4}$ radianes respecto al eje z, mientras que HTH es una rotación radianes respecto al eje x. Si componemos estas rotaciones, nos da el resultado:

$$T^2HT = cos^2\frac{\pi}{8}I - i(cos(\frac{\pi}{8}(X+Z)) + sin(\frac{\pi}{8}Y))sin(\frac{\pi}{8}).$$

Esto es una rotación a lo largo del eje $\vec{n} = (cos\frac{\pi}{8}, sin\frac{\pi}{8}, cos\frac{\pi}{8})$, con un ángulo θ que cumple $cos\frac{\theta}{2} = cos^2\frac{\pi}{8}$. Ahora, definimos $\delta > 0$, N un entero mayor a $\frac{2\pi}{\delta}$ y θ_k tal que $\theta_k = (k\theta)$ mod 2π. Claramente existen k y j tales que $\left|\theta_k - \theta_j\right| < \frac{2\pi}{N} < \delta$. Asumiendo que $k > j$, tenemos que $\left|\theta_{k-j}\right| < \delta$. A partir de esto es fácil ver que $\{\theta_{l(k-j)} | l \in \mathbb{Z}\}$ forma un conjunto del intervalo $[0, 2\pi)$ tal que cualquier elemento está a una distancia menor de δ. Por lo tanto, para todo $\epsilon > 0, \exists n$ tal que:

$$E(R_{\vec{n}}(\alpha), R_{\vec{n}}(\theta)^n) = E(R_{\vec{n}}(\alpha), R_{\vec{n}}(n\theta)) < \frac{\epsilon}{3}. \tag{4.2}$$

Por otra parte, también tenemos que $HR_{\vec{n}}(\theta)H = R_{\vec{m}}(\theta)$, siendo $\vec{m} = (cos\frac{\pi}{8}, -sin\frac{\pi}{8}, cos\frac{\pi}{8})$, de modo que:

$$E(R_{\vec{m}}(\theta), R_{\vec{m}}(n\theta)) < \frac{\epsilon}{3}. \tag{4.3}$$

Además, sabemos que por la proposición 4.1, para cualquier operador U, existen reales β, γ y φ tales que:

$$U = R_{\vec{n}}(\beta)R_{\vec{m}}(\gamma)R_{\vec{n}}(\varphi),$$

ignorando cierta fase global. De modo que usando la propiedad 4.2 y sustituyendo en esta las ecuaciones (4.2) y (4.3), obtenemos que existen naturales n_i, con $i = 1,2,3$ tales que:

$$\begin{aligned} E(U, R_{\vec{n}}(n_1\theta)R_{\vec{m}}(n_2\theta)R_{\vec{n}}(n_3\theta)) &= E(R_{\vec{n}}(\beta)R_{\vec{m}}(\gamma)R_{\vec{n}}(\varphi), R_{\vec{n}}(n_1\theta)R_{\vec{m}}(n_2\theta)R_{\vec{n}}(n_3\theta)) \\ &< E(R_{\vec{n}}(\beta), R_{\vec{n}}(n_1\theta)) + E(R_{\vec{m}}(\gamma), R_{\vec{m}}(n_2\theta)) + E(R_{\vec{n}}(\varphi), R_{\vec{n}}(n_3\theta)) < \epsilon. \end{aligned}$$

El hecho de que podamos construir las rotaciones anteriores con las puertas H,T,F y CNOT concluye la prueba. □

4.3. Complejidad computacional

En esta sección presentaremos el concepto de complejidad en el contexto de los problemas computacionales. Cuando queremos realizar cualquier tipo de cálculo con cualquier dispositivo de cómputo, es bastante útil tener una medida que nos dé una idea aproximada acerca de cuantos recursos utilizaremos al llevar a cabo dicho cálculo. En general se suelen utilizar medidas sobre el tiempo, como el número de operaciones necesarias, pero también puede usarse otras magnitudes como el espacio en memoria.

La idea principal de los estudios sobre complejidad de un algoritmo es abstraer el proceso general del mismo, obviando detalles del dispositivo que utilicemos para implementarlo. Es por esto que nos centramos en el coste abstracto de las operaciones, ignorando los tiempos o recursos específicos que utilizan distintos dispositivos para realizarlas. De este modo, obtenemos una medida general que depende únicamente del algoritmo, obviando detalles poco significativos de la implementación. Además, es obvio que el coste de un algoritmo siempre va a depender del tamaño de la tarea a realizar. Veamos algunos ejemplos.

Ejemplo 4.7. Supongamos que queremos estudiar la complejidad de un algoritmo que se base simplemente en sumar todos los números naturales de un vector de longitud n, cuya implementación en Python podría ser:

```
vector = [a_1,a_2,a_3,...,a_n] \\declaración explicita de un
sum = 0                         \\vector cualquiera de tamaño n.
for i in vector:
    sum = sum +i

return vector
```

Supongamos que el coste de las primeras asignaciones es una cierta constante m_1, mientras que el coste de la suma del bucle es m_2. De este modo, la función del coste $f(n) = m_1 + (n-1)m_2$, ya que realizamos $n-1$ sumas.

Ejemplo 4.8. Sea ahora una matriz cuadrada, con entradas naturales, de tamaño n. Imaginemos que usamos el código que se muestra a continuación para sumar todas sus entradas.

```
matrix = [[1,2,3,...,n],...,[1,2,3,...,n]] \\declaración explicita
sum = 0                                     \\de una matriz cualquiera nxn.
for i in matrix:
    sum = sum +i

return matrix
```

La complejidad de este algoritmo es, usando el mismo principio que en el ejemplo anterior $g(n) = q_1 + q_2(n-1)^2$, siendo q_1 y q_2 variables que dependen del dispositivo en el que se ejecute el código, entre otros factores.

En estos casos anteriores, siempre había un solo cauce en el flujo del programa o dicho en otras palabras, una única posibilidad para la ejecución. ¿Qué ocurre cuando hay más de una posibilidad? Supongamos que queremos medir la complejidad del siguiente programa.

```
vector = [a_1,a_2,a_3,...,a_n] \\declaración explicita de un
cte = m                         \\vector cualquiera de tamaño n.
for i in vector:
    if i > m:
        return i
return -1
```

¿Cuantas operaciones se llevan a cabo? Depende de donde esté el elemento del vector que supere a la constante m, si es que existe siquiera. Existen varios enfoques para lidiar con este problema, el principal es simplemente ponerse en el peor de los casos. En este código anterior, supondríamos que se da la posibilidad en la que el programa lleva a cabo la mayor cantidad de operaciones: no hay ningún elemento del vector mayor que m. Por otro lado, también

existen otras formas de medir, como centrarse en el mejor de los casos. Todo depende de si buscamos una cota inferior o superior para el coste.

Por otra parte, particularidades como el valor de las constantes m_1 y m_2, no son demasiado importantes, ya que dependen de propiedades específicas del lugar en el que se implementen los algoritmos. Lo esencial a la hora de medir la complejidad de un algoritmo es el *comportamiento asintótico*, es decir, como se comporta la función cuando hacemos el conjunto de datos de entrada cada vez más grande. Por ejemplo, las constantes del ejemplo 4.7, por alguna casualidad, pudieran ser mucho mayores que las del ejemplo 4.8, de modo que por ejemplo, para $n = 2$ $f(2) > g(2)$, pero nadie se atrevería a decir que el cálculo del primer ejemplo es más costoso que el del segundo. Esto se debe a que claramente existe un determinado n_0 tal que para todo $n > n_0$, $f(n) > g(n)$. Esta idea intuitiva nos lleva a una de las definiciones más importantes del estudio de la complejidad: la notación O.

Definición 4.4. Dada una función $f : \mathbb{N} \to \mathbb{Q}$ de complejidad de un algoritmo obtenida a través del enfoque del peor caso posible, entonces:

$$O(f) = \{g : \mathbb{N} \to \mathbb{Q} | \exists M, n_0 : |f(n)| < Mg(n), \forall n > n_0\}.$$

Esta definición puede resultar un tanto confusa, así que utilizaremos un ejemplo para presentar un uso práctico de la definición.

Ejemplo 4.9. Dada $f(n) = 12n^2 + 5n + 3$, $f \in O(n^2)$. Es fácil comprobar esto ya que $f(n) > 0, \forall n$, de modo que si elegimos $M = 50$ y $n_0 = 1$, entonces $f(n) < 50n^2, \forall n$.

Ejemplo 4.10. La función f del ejemplo 4.7 cumple que $f \in O(n)$. En este tipo de casos, se dice que el algoritmo tiene una complejidad *lineal*.

Ejemplo 4.11. La función g del ejemplo 4.8 cumple que $g \in O(n^2)$. En este caso, se dice que el algoritmo tiene una complejidad *polinomial cuadrática*.

Otra notación, que si bien inusual, también es importante es la notación Ω.

Definición 4.5. Dada una función $f : \mathbb{N} \to \mathbb{Q}$ de complejidad de un algoritmo obtenida a través del enfoque del mejor caso posible, entonces:

$$\Omega(f) = \{g : \mathbb{N} \to \mathbb{Q} | \exists M, n_0 : f(n) > Mg(n), \forall n > n_0\}.$$

La notación O nos permite establecer cotas superiores para la complejidad, mientras que la notación Ω nos permite establecer cotas inferiores. En general, utilizaremos la notación O u Ω para referirnos a la complejidad de soluciones a problemas, por lo que nos referiremos a la complejidad de un problema como la complejidad de su mejor solución.

La notación O es muy importante por ser la esencia de las *clases de complejidad*. Estas pueden definirse en función del tiempo (pasos de ejecución) o espacio (bits usados en el cómputo). Existe una teoría muy extensa en torno a este concepto, que involucran elementos como los lenguajes y las máquinas de Turing. Sin embargo, no trabajaremos con ellos puesto que seguiremos un enfoque superficial.

Una de las clases de complejidad más famosas es P, que es una abreviación de *polynomical time*. Una función de complejidad está en P si pertenece a $O(n^k)$ con $k \in \mathbb{N}$. Por extensión, un problema está en P si este tiene una solución cuya función de complejidad en tiempo esté en P. Por otro lado, tenemos la clase NP, que proviene de *non-deterministic polynomical time*. Un problema está en NP si las mejores soluciones del mismo pueden *verificarse* en tiempo polinómico. Claramente, esto tiene como consecuencia que $P \subset NP$. El hecho de que se de la igualdad es un problema abierto de extrema importancia, puesto que si esta se diera, esto implicaría que la complejidad de verificar un problema sería la misma que la de verificar una solución ya encontrada.

Por otro lado, tenemos $EXPTIME$, que como su nombre sugiere, es el conjunto de problemas que se resuelven en tiempo exponencial. Esto se describe una vez más con la notación O. Un problema está en $EXPTIME$ si las mejores soluciones tienen una complejidad $O(2^{p(n)})$, siento $p(n)$ una función polinómica.

También tenemos clases equivalentes para la complejidad en espacio. $PSPACE$ es el conjunto de problemas que tienen soluciones cuyo espacio necesario para ejecución aumenta de forma polinómica. Es decir, la función de complejidad respecto del espacio es $O(n^k)$ con $k \in \mathbb{N}$. La clase para espacio exponencial es $EXPSPACE$. Existen relaciones entre todas estas clases:

$$P \subset NP \subset PSPACE \subset EXPTIME \subset EXPSPACE.$$

Estas son las clases de complejidad más básicas. Sin embargo, aparece una nueva al incluir un factor probabilístico: la clase BPP. El nombre de esta clase proviene de *bounded-error probabilistic polynomical time*. Es la clase de problemas cuyas soluciones tienen complejidad polinómica en tiempo y además, cumplen la particularidad de que dichas soluciones son correctas con una probabilidad mínima de $\frac{1}{4}$.

Cabe preguntarse por qué nos iba a interesar una clase de problemas cuyas soluciones pueden ser incorrectas. Esto se debe a un resultado probabilístico que cuantifica este error y cuánto disminuye al repetir el algoritmo.

Proposición 4.2 (Cota de Chernoff)**.** *Supongamos las variables aleatorias independientes e igualmente distribuidas $X_1, \ldots, X_n$ que toman valores binarios y cumplen que $p_{X_i}(1) = \frac{1}{2} + \epsilon$ y $p_{X_i}(0) = \frac{1}{2} - \epsilon$, entonces se da que:*

$$p\left(\sum_{i=1}^{n} X_i \leq \frac{n}{2}\right) \leq e^{-2\epsilon^2 n}.$$

Ahora, supongamos un problema de decisión y un algoritmo que da una respuesta correcta al mismo con una probabilidad $\frac{1}{2} + \epsilon$ con $\epsilon > 0$. Utilizamos el siguiente criterio: tras ejecutar el algoritmo n veces, tomaremos la respuesta proporcionada que más veces ha aparecido. Ahora podemos utilizar la proposición anterior para ver que la probabilidad de que se dé 0 en la mitad o más de los casos, es decir, una respuesta incorrecta, disminuye *exponencialmente* cuando aumenta el número de ejecuciones. Esto nos dice que con una cantidad relativamente baja de repeticiones, podemos disminuir el error hasta dejarlo por debajo de cualquier cota, hasta incluso llegar a un punto en el que un fallo independiente al algoritmo sea más probable, como aquellos producidos por imperfecciones de componentes electrónicos o errores de redondeo.

Es fácil ver que $P \subset BPP$, la igualdad de nuevo es un problema abierto, pero se cree que esta igualdad debe darse, puesto que la cantidad de problemas que se creían en BPP pero no en P ha disminuido considerablemente a lo largo de los años. Un ejemplo de esto es el problema $PRIMES$, que consiste en ver si un número es primo o no y que se sabía que estaba en BPP, pero en el año 2002 se publico el *AKS primality test*, que es un algoritmo que en tiempo polinomial, puede dar con la solución [MA02]. Otro resultado importante es que $BPP \subset PSPACE$, cuya demostración no es trivial y no cubriremos en este trabajo.

A continuación, introducimos la computación cuántica en esta discusión: definimos la clase BQP o *bound-error quantum-probabilistic polynomical time*, que es el conjunto de problemas que tienen soluciones que se pueden implementar mediante un circuito cuántico con tamaño polinómico (entendiendo tamaño como pasos de ejecución) y que cumplen que dichas soluciones son correctas con una probabilidad mínima de $\frac{1}{4}$, es decir, es que el equivalente cuántico de BPP. Como ya hemos visto, un circuito cuántico puede simular cualquier circuito reversible, de modo que $BPP \subset BQP$. Sin embargo, no sabemos que $BQP = PSPACE$, que es otro problema abierto, pero sí que se da la inclusión, así que concluimos con la cadena de inclusiones $BPP \subset BQP \subset PSPACE$.

4.4. Algoritmos cuánticos

Ya tenemos todas las herramientas necesarias para la implementación de algoritmos en circuitos de índole cuántica. Hasta ahora hemos construido los cimientos teóricos. En la práctica, existen mayores limitaciones, pero siempre operaremos entorno a las siguientes suposiciones:

- Dentro de cualquier circuito cuántico, podemos contar con elementos tanto cuánticos como clásicos.
- Siempre trabajaremos con un sistema físico formado por n qubits.
- Podemos preparar cualquier estado inicial de trabajo que se encuentre en la base computacional.
- Podemos implementar cualquier puerta lógica cuántica con cualquier precisión, ya que hemos definido un conjunto universal de estas.
- Seremos capaces de realizar mediciones en la base computacional y por extensión, en cualquier base.

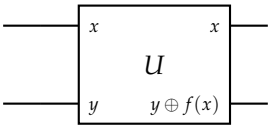
Las dos familias más importantes de algoritmos son aquella basadas en la *transformada cuántica de Fourier* y en el *algoritmo de búsqueda de Grover*. Sin embargo, empezaremos con un algoritmo mucho más simple que nos servirá como prueba de concepto para el uso de ordenadores cuánticos e introducir las numerosas ventajas de la computación cuántica respecto a la convencional.

4.4.1. Algoritmo de Deutsch-Jozsa

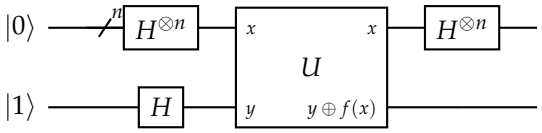
Este algoritmo si bien es simple, hace uso de dos propiedades fundamentales de la computación cuántica: la *interferencia* y la *paralelización*. Este algoritmo tiene como entrada una

función booleana $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ que o bien es constante o bien es balanceada. Una función balanceada es aquella que cumple que el número de valores que tienen como imagen el valor 1 es igual al número de aquellos que tienen como imagen el valor 0 . Este algoritmo es capaz de decirnos cuál de las dos posibilidades se dan con una sola evaluación de la función.

En primer lugar, tenemos que construir una llamada *caja negra* U_f. Una puerta lógica con entradas $x_1, \ldots, x_n, y$ capaz de calcular la biyección $|x_1, \ldots, x_n, y\rangle \to |y \oplus f(x_1, \ldots, x_n)\rangle$. La implementación de esta puerta no es importante. Su representación gráfica es:



A esta operación se le llama caja negra por el hecho de que no sabemos los detalles internos de la misma, pero sí los resultados que arroja. Esta puerta formará parte del siguiente circuito, que será el que utilizaremos para implementar el algoritmo:



Ahora, si preparamos la entrada como $|0\rangle^{\otimes n} |1\rangle$, al multiplicar por la entrada nos da:

$$(H^{\otimes n} \otimes H)(|0\rangle^{\otimes n} |1\rangle) = (H|0\rangle)^{\otimes n} \otimes H|1\rangle = \sum_{x \in \mathbb{Z}_2^n} \frac{|x\rangle}{\sqrt{2^n}} \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right).$$

A continuación aplicamos la puerta de que representa la operación U_f, lo que nos da:

$$\sum_{x \in \mathbb{Z}_2^n} \frac{|x\rangle}{\sqrt{2^n}} \left(\frac{|f(x)\rangle - |1 \otimes f(x)\rangle}{\sqrt{2}} \right).$$

Porque como ya hemos dicho, U_f solo afecta a y. Además, si $f(x) = 1$ entonces,$|f(x)\rangle - |1 \otimes f(x)\rangle = |1\rangle - |0\rangle$, mientras que si $f(x) = 0$, entonces $|f(x)\rangle - |1 \otimes f(x) = |0\rangle - |1\rangle\rangle$, por lo que podemos reescribir esta expresión como:

$$\sum_{x \in \mathbb{Z}_2^n} \frac{|x\rangle}{\sqrt{2^n}} \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) (-1)^{f(x)}.$$

Este es un claro ejemplo de *paralelización*: hemos aplicado la operación f a todas las secuencias de bits posibles, al mismo tiempo. Ahora aplicamos otra vez $H^{\otimes n}$ en x, que nos da:

$$H^{\otimes n} |x\rangle = \sum_{z \in \mathbb{Z}_2^n} \frac{(-1)^{x \cdot z} |z\rangle}{\sqrt{2^n}},$$

donde $\cdot$ es el producto escalar usual. Podemos insertar esta expresión en la anterior, resul-

tando:

$$\sum_{x\in\mathbb{Z}_2^n}\sum_{z\in\mathbb{Z}_2^n}\frac{(-1)^{x\cdot z+f(x)}\,|z\rangle}{2^n}\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right).$$

Con el paso anterior hemos aplicado *interferencia* entre los distintos estados posibles. La parte correspondiente a x nos dice la amplitud de cada posibilidad para x. En particular, si $x=0,\ldots,0$, la amplitud de este estado es $\sum_{x\in\mathbb{Z}_n^2}\frac{(-1)^{f(x)}}{2^n}$. Si f es constante, entonces dicha amplitud es $+1$ o -1. Pero como la probabilidad de conseguir este estado es el módulo de su amplitud, si f es constante, el hecho de obtener todo 0 es un hecho seguro. Por otro lado, si f es balanceada, entonces todos los sumandos se anulan, obteniendo una amplitud nula, por lo que si la función es balanceada, la probabilidad de medir todo 0 es nula. Así, concluimos que si al medir f obtenemos todos 0, entonces es constante y si hay algún 1, entonces es balanceada.

Hemos obtenido una propiedad global de f con una sola evaluación de la misma. En un ordenador clásico, habríamos necesitado como mínimo $2^{n-1}+1$ evaluaciones para concluir la misma propiedad. Si bien este algoritmo en general no es demasiado útil, sirve como un buen ejemplo, puesto que es el algoritmo más simple en el que se demuestra como un ordenador cuántico puede superar en rendimiento a uno clásico.

4.4.2. Algoritmos basados en la transformada cuántica de Fourier

En primer lugar, definimos la transformación que jugará un papel central para toda una familia de algoritmos.

Definición 4.6. Sea $z\in\mathbb{C}^N$ un vector complejo. Definimos la *transformada discreta de Fourier* como el operador que realiza la aplicación:

$$(x_0,\ldots,x_{N-1})\to(y_0,\ldots,y_{N-1}),$$

donde cada y_k se define como:

$$y_k:=\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}x_j e^{\frac{2\pi ijk}{N}}.$$

Ahora, con la notación que hemos usado hasta ahora, si tenemos un vector complejo en la base computacional expresado como:

$$|\psi\rangle=\sum_{j=0}^{N-1}x_j\,|j\rangle\,,$$

entonces la transformada discreta de Fourier aplicada a $|\psi\rangle$ nos da la aplicación:

$$TFD(|\psi\rangle)=TFD\left(\sum_{j=0}^{N-1}x_j\,|j\rangle\right)=\sum_{k=0}^{N-1}y_k\,|k\rangle=\frac{1}{\sqrt{N}}\sum_{k=0}^{N-1}\sum_{j=0}^{N-1}x_j e^{\frac{2\pi ijk}{N}}\,|k\rangle.$$

A partir de ahora, siempre consideraremos $N=2^n$ para algún n, puesto que en general trabajaremos con productos de n qubits. A continuación, veremos como transformar la

imagen por la transformada discreta de Fourier en otra expresión más manejable para su implementación. Partimos de la imagen para un vector unitario de la base computacional.

$$TFD(|j\rangle) = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{\frac{2\pi ijk}{2^n}} |k\rangle.$$

Si cambiamos el número natural k por su expresión binaria $k_1, \ldots, k_n$ obtenemos:

$$TFD(|j\rangle) = \frac{1}{\sqrt{2^n}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} e^{2\pi ij(\sum_l^n k_l 2^{-l})} |k_1 \ldots k_n\rangle.$$

Ahora, podemos expresar $|k_1 \ldots k_n\rangle$ como producto de Kronecker de $|k_1\rangle \ldots |k_n\rangle$ y si además agrupamos cada sumando de la suma de fracciones $\sum_{l=1}^n k_l 2^{-l}$ con su respectivo $|k_l\rangle$ tenemos:

$$\frac{1}{\sqrt{2^n}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} \bigotimes_{l=1}^{n} e^{2\pi ijk_l 2^{-l}} |k_l\rangle.$$

Ahora simplemente tenemos que reagrupar los términos de la suma para agruparlos dentro de los productos de Kronecker, llegando a una expresión más condensada.

$$\frac{1}{\sqrt{2^n}} \bigotimes_{l=1}^{n} \left(\sum_{k_l=0}^{1} e^{2\pi ijk_l 2^{-l}} |k_l\rangle \right) = \frac{1}{\sqrt{2^n}} \bigotimes_{l=1}^{n} \left(|0\rangle + e^{2\pi ijk_l 2^{-l}} |1\rangle \right)$$

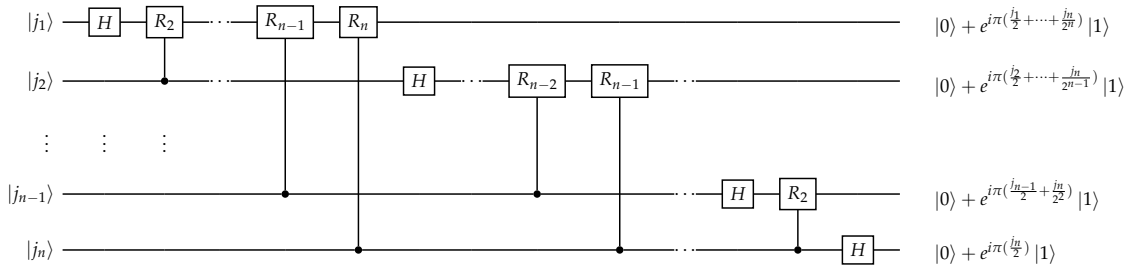
Si desarrollamos el producto de Kronecker, obtenemos:

$$\frac{\left(|0\rangle + e^{2ij\pi \frac{k_n}{2}} |1\rangle \right) \otimes \cdots \otimes \left(|0\rangle + e^{2i\pi j(\frac{k_1}{2} + \cdots + \frac{k_n}{2^n})} |1\rangle \right)}{\sqrt{2^n}}. \tag{4.4}$$

En base a esto, definimos una nueva puerta lógica:

$$R_k := \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^k}} \end{bmatrix}, k \geq 2.$$

Esta puerta es una generalización de las puertas 'pi octavos', $\frac{\pi}{8} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$ y fase $F = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$.
Ahora veremos que podemos utilizar esta puerta para calcular la transformada discreta de Fourier.



Analicemos este circuito. Al ser la entrada $|j_1 \dots j_n\rangle$, cuando aplicamos la puerta de Hadamard al primer qubit, obtenemos una expresión u otra en función de que este qubit sea $|0\rangle$ o $|1\rangle$:

$$\frac{1}{\sqrt{2}}(|0\rangle + (-1)^{j_1}|1\rangle)|j_2 \dots j_n\rangle = \frac{1}{\sqrt{2}}(|0\rangle + (e^{i\pi})^{j_1}|1\rangle)|j_2 \dots j_n\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi\frac{j_1}{2}}|1\rangle)|j_2 \dots j_n\rangle .$$

Ahora aplicamos la puerta R_2 condicionada al qubit $|j_2\rangle$. Esta puerta deja el estado la componente $|0\rangle$ inalterada, mientras que multiplica por $e^{\frac{2\pi i}{2^2}}$ la amplitud de $|1\rangle$. Como está condicionada, podemos expresar R_2 como multiplicar por $e^{\frac{2\pi i j_2}{2^2}}$, de modo que el estado nos queda:

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi(\frac{j_1}{2}+\frac{j_2}{4})}|1\rangle)|j_2 \dots j_n\rangle$$

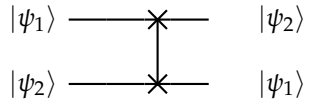
Al aplicar todas las puertas R_j condicionales restantes, obtenemos el estado:

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi(\frac{j_1}{2}+\cdots+\frac{j_n}{2^n})}|1\rangle)|j_2 \dots j_n\rangle .$$

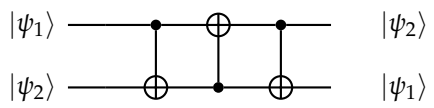
Si aplicamos este mismo proceso al resto de qubits de la entrada, de forma que para cada qubit $|j_q\rangle$, con $q = 1, \dots, n-1$, usamos la composición de puertas condicionales R_2 hasta R_{n-q+1}, mientras que para el último solo aplicamos la puerta Hadamard. De este modo, la salida del circuito quedaría:

$$\frac{\left(|0\rangle + e^{2i\pi(\frac{k_1}{2}+\cdots+\frac{k_n}{2^n})}|1\rangle\right) \otimes \cdots \otimes \left(|0\rangle + e^{2i\pi\frac{k_n}{2}}|1\rangle\right)}{\sqrt{2^n}}.$$

Sin embargo, esta expresión no es exactamente la misma que la que hemos calculado en la expresión (4.4), ya que está al revés. Pero esto no es problema, ya que podemos utilizar varias puertas SWAP que se definen como:



Esta puerta puede implementarse como:



Podemos componer varias de estas puertas para permutar totalmente la salida, llegando a la expresión que describe la transformada cuántica de Fourier como:

$$|k_1 \dots k_n\rangle \rightarrow \frac{\left(|0\rangle + e^{2i\pi \frac{k_n}{2}} |1\rangle\right) \otimes \cdots \otimes \left(|0\rangle + e^{2i\pi(\frac{k_1}{2} + \cdots + \frac{k_n}{2^n})} |1\rangle\right)}{\sqrt{2^n}}. \tag{4.5}$$

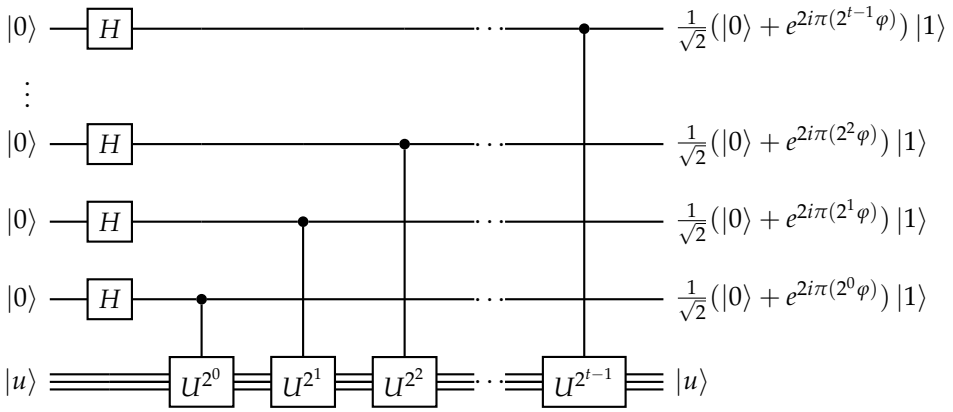
Cabe destacar que , al estar este circuito formado por transformaciones unitarias, hemos demostrado indirectamente que la transformada discreta de Fourier se trata de una transformación unitaria.

4.4.2.1. Estimación de fase

La estimación de fase es un algoritmo que se utiliza como subrutina en otros algoritmos de grandísima utilidad que veremos a continuación. La estimación de fase tiene como entrada una matriz U, con un autovalor $|u\rangle$. Ahora, suponemos que el autovalor asociado a este autovector es $e^{2\pi i\varphi}$, con $\varphi \in \mathbb{R}$ que a priori no sabemos y que además podemos implementar como caja negra el operador U^{2^j}, para un j positivo. El objetivo de este algoritmo es estimar φ.

Utilizaremos dos registros de qubits. En el primero, almacenaremos t qubits prefijados a $|0\rangle$. El número t es un parámetro del algoritmo que definirá tanto la precisión de la estimación como la probabilidad de que esta sea correcta. En el segundo registro, almacenaremos la representación binaria de la etiqueta u.

El primer paso del algoritmo es aplicar el siguiente circuito a ambos registros, suponiendo que el primero está compuesto por t qubits prefijados a $|0\rangle$:



Para cada pareja $|1\rangle \otimes |u\rangle$, si aplicamos la operación condicional y al ser $|u\rangle$ un autovector

con autovalor λ, el resultado de $U^{2^j}|u\rangle = e^{2\pi i\varphi 2^j}|u\rangle$, por lo que $|1\rangle \otimes U^{2^j}|u\rangle$ puede escribirse como $e^{2\pi i\varphi 2^j}|1\rangle \otimes |u\rangle$. Por lo tanto, en primer lugar tenemos que al aplicar la puerta de Hadamard en cada qubit del primer registro, obtenemos que el estado del sistema compuesto es:

$$\frac{(|0\rangle + |1\rangle) \otimes \cdots \otimes (|0\rangle + |1\rangle))}{\sqrt{2^t}}.$$

Por lo tanto, ahora podemos aplicar cada una de las operaciones condicionales como acabamos de explicar, dándonos el resultado del primer registro del circuito.

$$\frac{(|0\rangle + e^{2i\pi 2^{t-1}\varphi}|1\rangle) \otimes \cdots \otimes (|0\rangle + e^{2i\pi\varphi}|1\rangle)}{\sqrt{2^t}}.$$

Ahora, supongamos que φ puede expresarse exactamente en t bits como el valor decimal $\varphi = \frac{\varphi_1}{2} + \frac{\varphi_2}{2^2} + \cdots + \frac{\varphi_t}{2^t}$, con $\varphi_i \in \mathbb{Z}^2$, ya que la fase tiene periodo 2π. De modo que podemos reescribir la ecuación anterior como:

$$\frac{(|0\rangle + e^{2i\pi 2^{t-1}(\frac{\varphi_1}{2}+\frac{\varphi_2}{2^2}+\cdots+\frac{\varphi_t}{2^t})}|1\rangle) \otimes \cdots \otimes (|0\rangle + e^{2i\pi(\frac{\varphi_1}{2}+\frac{\varphi_2}{2^2}+\cdots+\frac{\varphi_t}{2^t})}|1\rangle)}{\sqrt{2^t}} =$$

$$\frac{(|0\rangle + e^{2i\pi(2^{t-2}\varphi_1+2^{t-3}\varphi_3+\cdots+\frac{\varphi_t}{2})}|1\rangle) \otimes \cdots \otimes (|0\rangle + e^{2i\pi(\frac{\varphi_1}{2}+\frac{\varphi_2}{2^2}+\cdots+\frac{\varphi_t}{2^t})}|1\rangle)}{\sqrt{2^t}} =$$

$$\frac{(|0\rangle + e^{2i\pi 2^{t-2}\varphi_1}e^{2i\pi 2^{t-3}\varphi_2}\ldots e^{2i\pi\frac{\varphi_t}{2}}|1\rangle) \otimes \cdots \otimes (|0\rangle + e^{2i\pi(\frac{\varphi_1}{2}+\frac{\varphi_2}{2^2}+\cdots+\frac{\varphi_t}{2^t})}|1\rangle)}{\sqrt{2^t}} =$$

$$\frac{(|0\rangle + e^{2i\pi\frac{\varphi_t}{2}}|1\rangle) \otimes \cdots \otimes (|0\rangle + e^{2i\pi(\frac{\varphi_1}{2}+\frac{\varphi_2}{2^2}+\cdots+\frac{\varphi_t}{2^t})}|1\rangle)}{\sqrt{2^t}}.$$

Podemos ver cómo esta expresión es convenientemente igual a la salida de la transformada discreta de Fourier descrita en (4.5). Por lo tanto, al ser esta una transformación unitaria, tiene inversa que podemos aplicar al primer registro, obteniendo:

$$\frac{(|0\rangle + e^{2i\pi\frac{\varphi_t}{2}}|1\rangle) \otimes \cdots \otimes (|0\rangle + e^{2i\pi(\frac{\varphi_1}{2}+\frac{\varphi_2}{2^2}+\cdots+\frac{\varphi_t}{2^t})}|1\rangle)}{\sqrt{2^t}} \to |\varphi_1 \ldots \varphi_t\rangle$$

Al realizar una simple medición del primer registro, obtenemos la expresión binaria de φ, terminando así el algoritmo. Sin embargo, hemos supuesto que φ podía escribirse de forma exacta con t bits. Esto no tendría por qué darse. Pero aun así, puede demostrarse que este algoritmo puede producir una estimación con precisión arbitraria eligiendo un t adecuado. Esta estimación viene dada por la siguiente proposición.

Proposición 4.3. *En el algoritmo de estimación de fase, el elegir un número de bits de precisión:*

$$t = n + \left(\log\left(2 + \frac{1}{2\epsilon}\right)\right)$$

nos da una probabilidad de $1 - \epsilon$ *de obtener una estimación de la fase,* φ*, exacta hasta el n-ésimo bit.*

Por otra parte, al principio del capítulo hemos definido las herramientas que podemos utilizar para la construcción de algoritmos cuánticos. Una de ellas es la capacidad para preparar cualquier estado inicial en la base computacional. Sin embargo, esto no se extiende

a todos los estados posibles. Por tanto, no sabemos si es posible preparar el estado inicial $|u\rangle$. Sin embargo, podemos preparar un estado conocido $|\psi\rangle$ cualquiera. Este estado puede escribirse en función de los autovectores de cualquier operador, puesto que estos forman una base. De este modo $|\psi\rangle = \sum_u c_u |u\rangle$, tal que cada $|u\rangle$ tiene asociado un autovalor con fase φ_u. Si ejecutamos el algoritmo de estimación de fase con $|0\dots0\rangle \sum_u c_u |u\rangle$ como entrada, produciremos $\sum_u c_u |\tilde{\varphi_u}\rangle |u\rangle$, siendo $|\tilde{\varphi_u}\rangle$ una estimación de φ_u. Por lo tanto, si medimos la salida obtendremos cada $|\tilde{\varphi_u}\rangle$, con probabilidad $|c_u|^2$. Esto produce un nuevo factor que afecta al error, ya que existe la posibilidad de que midamos un $|\tilde{\varphi_u}\rangle$ que no se corresponda a la estimación para la fase que buscábamos. En estas condiciones, si elegimos t como en la proposición 4.3, el error aumenta $|c_u|^2(1-\epsilon)$.

El procedimiento de este algoritmo puede resumirse en los siguientes pasos:

1. Estado inicial, $|0\rangle^{\otimes t} |u\rangle$, donde $t = n + \log\left(1 + \frac{1}{2\epsilon}\right)$.
2. Creamos la superposición $\frac{1}{2^t} \sum_{j=0}^{2^t-1} |j\rangle |u\rangle$.
3. Aplicamos la operación caja negra $\frac{1}{2^t} \sum_{j=0}^{2^t-1} |j\rangle U^j |u\rangle = \frac{1}{2^t} \sum_{j=0}^{2^t-1} e^{2i\pi j\varphi_u} |j\rangle |u\rangle$.
4. Aplicamos la transformada inversa de Fourier, resultando $|\tilde{\varphi}_u\rangle |u\rangle$.
5. Medimos el primer registro, obteniendo $\tilde{\varphi}_u$, que es una estimación de φ_u con una precisión de n bits, con probabilidad $1-\epsilon$.

Alternativamente, podemos utilizar la entrada $|0\rangle^{\otimes t} |\psi\rangle$, siendo $|\psi\rangle$ un estado cualquiera conocido, cuando no conozcamos el autovector u. En este caso, la probabilidad de éxito baja a $|c_u|(1-\epsilon)$, siendo c_u la amplitud de $|u\rangle$ en la superposición que forma $|1\rangle$.

4.4.2.2. Orden de un natural

Dados números naturales cualesquiera x y N, se define el orden de x módulo N como el menor número r tal que $x^r = 1(\mod N)$. Definimos L como la parte entera de $\log_2(N)$, es decir, el número de bits necesarios para definir N. El algoritmo para el cálculo de orden es únicamente el algoritmo de estimación usando como operador:

$$U |y\rangle = |xy \mod N\rangle .$$

con $y \in \mathbb{Z}_2^L$. Ahora, definimos el estado:

$$|u_s\rangle := \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{\frac{-2\pi isk}{r}} \left| x^k \mod N \right\rangle ,$$

para enteros $0 \leq s \leq r-1$. Podemos ver que son autovalores de U con un simple cálculo:

$$U |u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{\frac{-2\pi isk}{r}} \left| x^{k+1} \mod N \right\rangle = e^{\frac{2\pi is}{r}} |u_s\rangle .$$

Usando el algoritmo de estimación de fase podemos hallar los respectivos autovalores $e^{\frac{2\pi is}{r}}$, a partir de los cuales podemos calcular r. Sin embargo, existe un problema: ¿Cómo vamos a

preparar el estado $|u_s\rangle$ si no conocemos r? Podemos rodear este obstáculo haciendo uso del siguiente cálculo:

$$\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}|u_s\rangle = \frac{1}{r}\sum_{s=0}^{r-1}\sum_{k=0}^{r-1} e^{\frac{-2\pi isk}{r}}\left|x^k \mod N\right\rangle.$$

Si cambiamos el orden de las sumatorias, obtenemos:

$$\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}|u_s\rangle = \frac{1}{r}\sum_{k=0}^{r-1}\sum_{s=0}^{r-1} e^{\frac{-2\pi isk}{r}}\left|x^k \mod N\right\rangle.$$

Ahora, separamos la expresión en el término $k = 0$ y los demás.

$$\begin{aligned}\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}|u_s\rangle &= \frac{1}{r}\sum_{s=0}^{r-1} e^{\frac{-2\pi is0}{r}}\left|x^0 \mod N\right\rangle + \frac{1}{r}\sum_{k=1}^{r-1}\sum_{s=0}^{r-1} e^{\frac{-2\pi isk}{r}}\left|x^k \mod N\right\rangle \\ &= |1\rangle + \frac{1}{r}\sum_{k=1}^{r-1}\sum_{s=0}^{r-1} e^{\frac{-2\pi isk}{r}}\left|x^k \mod N\right\rangle.\end{aligned}$$

Para cada k la sumatoria en la variable s es una serie geometrica, por lo que sabemos que:

$$\sum_{s=0}^{r-1} e^{\frac{-2\pi isk}{r}} = \frac{1-e^{-2\pi ik}}{1-e^{-2\pi i\frac{k}{r}}} = 0.$$

Por lo que finalmente obtenemos:

$$\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}|u_s\rangle = |1\rangle.$$

De este modo, podemos introducir como entrada simplemente el estado $|1\rangle$ y al medir la salida, mediremos una estimación de una de las fases $\frac{s}{r}$, siendo s algún número entre 0 y $r-1$. A partir de este número, basta con expresarlo como una fracción irreducible. Podrían ocurrir varios casos en los que este último paso no sea tan sencillo, como el caso en el que $s = 0$ o que s y r no sean coprimos. Sin embargo estos casos son los menos probables. La probabilidad de obtener $s = 0$ es $\frac{1}{r}$. Mientras que el número de primos menores que un r dado es como mínimo $\frac{r}{2\log r}$, por lo que la probabilidad de que s sea primo es $\frac{1}{2\log r} < \frac{1}{2\log N}$. Por lo tanto, si repetimos el algoritmo $2\log N$ veces, obtenemos una probabilidad mayor a $\frac{1}{2}$ de obtener un decimal cuya fracción irreducible nos de r.

De forma más esquemática, el algoritmo queda:

1. Utilizamos como entrada $|0\rangle^{\otimes t}|1\rangle^{\otimes L}$, siendo L los bits necesarios para describir N y $t = 2L + 1 + \log\left(2 + \frac{1}{2\epsilon}\right)$.
2. Aplicamos el algoritmo de estimación de fase.
3. En el primer registro formado por t qubits, queda una superposición de estimacines de fase $\left|\frac{\tilde{s}}{r}\right\rangle$, siendo r el orden buscado y s un número natural menor que r.
4. Al medir, obtenemos un determinado estimador $\frac{s_0}{r}$, preciso hasta L bits, con probabili-

dad $1 - \epsilon$.

5. A partir de la fracción irreducible, obtenemos r.

4.4.2.3. El algoritmo de Shor para factorización en primos

El último algoritmo que mostraremos, también será el más importante, puesto que tiene implicaciones prácticas enormes. El factorizar un número natural siempre ha sido un problema computacionalmente difícil, por lo que muchos métodos criptográficos están basados en esta tarea. El mejor algoritmo para la factorización es el denominado *General number field sieve*, que tiene complejidad exponencial. Por el contrario, el algoritmo cuántico que presentaremos tiene complejidad $O(log(N)^3)$, lo que pone una mejora exponencial. Este algoritmo trata de una adaptación del cálculo del orden de un natural. Para esta adaptación, utilizaremos los siguientes enunciados.

Teorema 4.5. *Sea N un número natural compuesto que puede representarse con L bits, mientras que x es una solución no trivial de la ecuación $x^2 = 1 \mod N$ (es decir, x no es ni $1 \mod N$ ni $-1 \mod N$). Entonces se cumple que $mcd(x-1, N)$ o $mcd(x+1, N)$ es un divisor de N.*

Teorema 4.6. *Supongamos que N se factoriza como $N = p_1^{\alpha_1} \dots p_m^{\alpha_m}$. Dado un número aleatorio $1 \leq x \leq N-1$ tal que x no tenga factores comunes con N. Si r es el orden de $x \mod N$, entonces:*

$$p(r \text{ sea par y } \sqrt{x^r} \neq -1 \mod N) \geq 1 - \frac{1}{2^m}.$$

Proposición 4.4. *Existe un algoritmo capaz de comprobar si un número natural N expresado por L bits puede expresarse como a^b con $a > 1$ y $b \geq 2$ con complejidad $O(L^3)$.*

Demostración. En primer lugar, es fácil ver que b es como mucho L, puede verse que si el valor más bajo para a es 2 y $2^{L+1} > N$. A continuación, para cada $b \leq L$:

1. Calculamos $x = \frac{L}{b}$ y los enteros u_1 y u_2 más cercanos a 2^x (enteros inmediatamente superior e inferior). Este proceso tiene una complejidad de $O(L^2)$.
2. Calculamos u_1^b y u_2^b y comprobamos si alguno es N, de nuevo, con una complejidad $O(L^2)$.

Este bucle se repite un número máximo de $L = \log_2(N)$, por lo que el algoritmo tiene una complejidad de $O(L^3)$. □

Con estas herramientas, ya estamos en posición de desarrollar el algoritmo de factorización:

1. Comprobamos si N es par, en cuyo caso devolvemos el factor 2.
2. Comprobamos si $N = a^b$. Si se da, devolvemos a.
3. Elegimos un número aleatorio entre 1 y $N-1$. Si $mcd(x, N) > 1$ entonces devolvemos $mcd(x, N)$ como factor.
4. Utilizamos el algoritmo de cálculo de orden para hallar r tal que $x^r = 1 \mod N$.
5. Si r es par, y $\sqrt{x^r} \neq -1 \mod N$, entonces calculamos $mcd(\sqrt{x^r} - 1, N)$ y $mcd(\sqrt{x^r} + 1, N)$, uno de los cuales será un factor de N por el teorema 4.5. Si no, se dan las primeras dos condiciones, el algoritmo devuelve error.

En cada paso, el algoritmo nos devuelve uno de los factores de N, si dividimos N entre ese factor e iteramos hasta que $N = 1$, obtenemos la factorización de N.

4.4.3. Algoritmos basados en el algoritmo de búsqueda de Grover

4.4.3.1. Algoritmo de búsqueda básico

Un problema de búsqueda es una situación en la que se desea encontrar una solución dentro de un conjunto de posibles estados o configuraciones, mediante la exploración sistemática de ese conjunto llamado espacio de búsqueda. Estos son problemas en los que las soluciones no se calculan, sino que se encuentran entre otras muchas posibilidades.

Lo primero que haremos será definir la manera en la que reconocemos las soluciones del problema. Supongamos que tenemos un espacio de búsqueda de tamaño N que representamos con n bits y enumeramos cada una de las soluciones con un índice x que va de 0 a $N-1$, siendo M el número total de soluciones. Supongamos que tenemos una función f que tiene por dominio el conjunto de índices del espacio de búsqueda del problema y tiene como salida 1 la entrada se está asociada a una solución y 0 en caso contrario. Definimos el *oráculo* como una caja negra capaz de realizar el cálculo:

$$|x\rangle\,|q\rangle \rightarrow |x\rangle\,|q \oplus f(x)\rangle\,.$$

Si definimos el oráculo de esta manera, es claro que si $|q\rangle = |0\rangle$, entonces vemos que su acción es equivalente a realizar la acción:

$$|x\rangle\,|0\rangle \rightarrow |x\rangle\,|f(x)\rangle\,.$$

Sin embargo, si aplicamos el oráculo a $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$, entonces:

$$|x\rangle\,\frac{|0\rangle - |1\rangle}{\sqrt{2}} \rightarrow |x\rangle\left(\frac{|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}}\right).$$

Es posible reescribir esta transformación como:

$$|x\rangle\,\frac{|0\rangle - |1\rangle}{\sqrt{2}} \rightarrow |x\rangle\,(-1)^{f(x)}\,\frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

Como el oráculo no tiene ningún efecto sobre el primer registro, que contiene el índice x, podemos obviarlo, condensando su acción en la operación:

$$|x\rangle \rightarrow |x\rangle\,(-1)^{f(x)}.$$

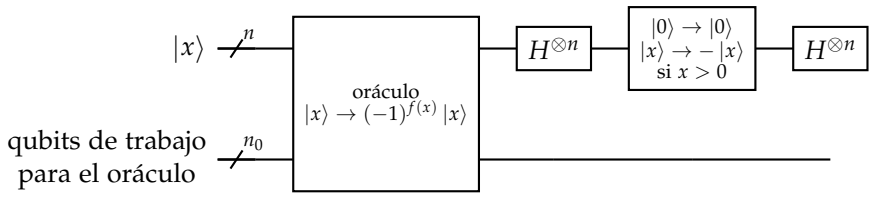
La idea principal es intentar encontrar la solución con el menor número posible de aplicaciones del oráculo.

En primer lugar, tomamos como entrada un estado en el que todos los qubits son $|0\rangle$, es decir $|0\rangle^{\otimes n}$. Podemos aplicar la puerta de Hadamard para generar una superposición de todos los estados posibles, generando el estado:

$$|\psi\rangle = \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle\,.$$

Ahora, hacemos uso de una subrutina que constituye el esqueleto de del algoritmo: la

iteración de Grover. Este procedimiento viene dado por el siguiente circuito.



La operación que aplicamos sobre $|x\rangle$ después del oráculo es aplicar la puerta de Hadamard, una transformación de amplitudes para todas las componentes de la superposición excepto $|0\dots0\rangle$ y despues la Hadamard de nuevo. La transformación de las amplitudes puede verse como multiplicar por la matriz $2|0\rangle\langle 0| - I$. De modo que tenemos $H^{\otimes n}(2|0\rangle\langle 0| - I)H^{\otimes n} = H^{\otimes n}2|0\rangle\langle 0|H^{\otimes n} - H^{\otimes n}IH^{\otimes n} = 2|\psi\rangle\langle\psi| - I$. Así que podemos concluir que, si O es el operador del oráculo sobre $|x\rangle$, este circuito puede describirse como la operación $G = O(2|\psi\rangle\langle\psi| - I)$.

Veamos ahora como utilizar este procedimiento. Si S es el espacio de búsqueda, definamos los conjuntos:

$$A = \{x \in S : f(x) = 1\},$$
$$B = \{x \in S : f(x) = 0\}$$

y los estados siguientes, siendo M el número de soluciones del algoritmo de búsqueda:

$$|\alpha\rangle = \frac{1}{\sqrt{N-M}} \sum_{x\in A} |x\rangle$$
$$|\beta\rangle = \frac{1}{\sqrt{M}} \sum_{x\in B} |x\rangle .$$

Que además son claramente perpendiculares. Es fácil ver que $A + B = S$, luego si $|\psi\rangle$ es la superposición de todos los estados posibles, entonces puede escribirse como:

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x\in S} = \frac{1}{\sqrt{N}} \sum_{x\in A} + \frac{1}{\sqrt{N}} \sum_{x\in B} = \sqrt{\frac{N-M}{N}} |\alpha\rangle + \sqrt{\frac{M}{N}} |\beta\rangle .$$

Habíamos definido G como la combinación de las operaciones O y $2|\psi\rangle\langle\psi| - I$. Aplicar la primera sobre $|\psi\rangle$ nos da:

$$O(|\psi\rangle) = O\left(\sqrt{\frac{N-M}{N}} |\alpha\rangle + \sqrt{\frac{M}{N}} |\beta\rangle\right) = \sqrt{\frac{N-M}{N}} |\alpha\rangle - \sqrt{\frac{M}{N}} |\beta\rangle .$$

Esta operación se trata de una simetría respecto de $|\alpha\rangle$ en el plano formado por $|\alpha\rangle$ y $|\beta\rangle$. Por otro lado, la operación $2|\psi\rangle\langle\psi| - I$ representa una rotación en el mismo plano respecto del vector $|\psi\rangle$. Esto puede comprobarse fácilmente si tomamos un vector del plano, $|\phi\rangle$ tal que $\langle\psi|\phi\rangle = 0$, entonces cualquier vector del plano puede expresarse como $a|\psi\rangle + b|\phi\rangle$ con

$a, b \in \mathbb{C}$ de modo que si tomamos un $|v\rangle = a\,|\psi\rangle + b\,|\psi\rangle$ cualquiera de este plano:

$$(2\,|\psi\rangle\,\langle\psi| - I)v = (2\,|\psi\rangle\,\langle\psi| - I)(a\,|\psi\rangle + b\,|\phi\rangle) = a\,|\psi\rangle - b\,|\phi\rangle\,.$$

Lo que representa el efecto producido por una simetría respecto del vector $|\psi\rangle$ en el plano formado por $|\alpha\rangle$ y $|\beta\rangle$ y, por tanto, el operador G se trata de una rotación en dicho plano. De hecho, si definimos $\theta \in \mathbb{R}$ tal que $\cos\frac{\theta}{2} = \sqrt{\frac{N-M}{N}}$, entonces podemos escribir $|\psi\rangle$ como:

$$|\psi\rangle = \cos\frac{\theta}{2}\,|\alpha\rangle + \sin\frac{\theta}{2}\,|\beta\rangle\,.$$

A partir de aquí, el resultado de aplicar G queda:

$$G\,|\psi\rangle = \cos\frac{3\theta}{2}\,|\alpha\rangle + \sin\frac{3\theta}{2}\,|\beta\rangle\,.$$

De este modo puede verse que para un $k \geq 1$ arbitrario:

$$G^k\,|\psi\rangle = \cos\left(\frac{2k+1}{2}\theta\right)|\alpha\rangle + \sin\left(\frac{2k+1}{2}\theta\right)|\beta\rangle\,.$$

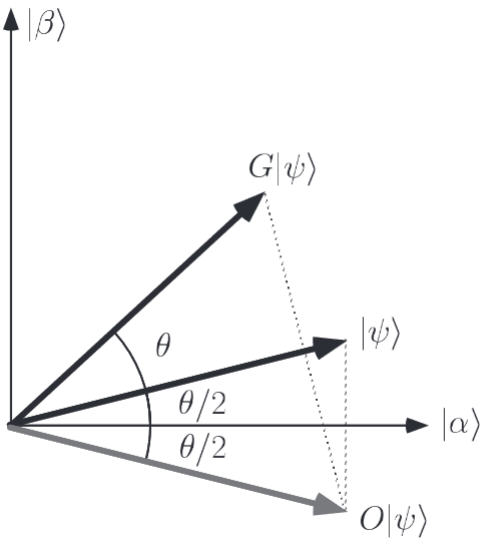


Figura 4.3: Efecto del operador G en el plano formado por $|\alpha\rangle$ y $|\beta\rangle$

Por ende, si elegimos apropiadamente en función de N y M, podemos encontrar un k_0 tal que $\sin\left(\frac{2k_0+1}{2}\theta\right) \approx 1$. Esto produce un efecto muy conveniente: al realizar una medición de $G^{k_0}\,|\psi\rangle$ obtenemos con altísima probabilidad uno de los vectores de la superposición que forman $|\beta\rangle$, que es uno de los elementos del espacio de búsqueda que buscábamos. El número de veces que tendremos que repetir la iteración de Grover no puede saberse a ciencia cierta, pero si que crece exponencialmente. De forma más específica, el número de iteraciones

es $O(\sqrt{2^n})$, por otro lado, también existe una cota inferior que nos permite medir $|\beta\rangle$ con probabilidad mayor a $\frac{M}{N}$, que es el número natural más cercano a la expresión $\frac{\arccos\sqrt{\frac{M}{N}}}{\theta}$. De la misma manera, también existe una cota superior, mucho más simple, que será la que utilicemos usualmente. Dicha cota es el entero más cercano a la expresión:

$$\frac{\pi}{4}\sqrt{\frac{M}{N}}.$$

Sin embargo, estas aproximaciones solo son efectivas cuando la fracción $\frac{M}{N}$ es menor que $\frac{1}{2}$. Podemos sobrepasar este obstáculo simplemente doblando el tamaño del espacio de búsqueda de N a $2N$. Esto produce una probabilidad de éxito menor a $\frac{1}{2}$, lo que nos permite aplicar el algoritmo varias veces para producir un acierto casi seguro.

El proceso iterativo anteriormente descrito es la esencia del *algoritmo del búsqueda de Grover*, que esquematizamos a continuación.

Supongamos que tenemos un problema de búsqueda con una única solución, x_0, de entre 2^n posibilidades y f una función tal que $f(x_0) = 1$ e igual a 0 en otro caso. El algoritmo se desarrolla de la siguiente manera:

1. Tenemos como un entrada el estado $|0\rangle^{\otimes n}|0\rangle$.
2. Aplicamos la puerta $H^{\otimes n}$ a los primeros n qubits y HX al último, obteniendo el estado $\frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$.
3. Aplicamos G^k al estado anterior, siendo G el operador de Grover y k un natural apropiado dependiente de las características específicas del problema, lo que nos da una estimación $|x_0\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$.
4. Medimos en los primeros n qubits, obteniendo x_0.

4.4.3.2. Algoritmo de conteo cuántico

En el algoritmo anterior pudimos ver que para obtener un número de iteraciones preciso para la iteración de Grover, era necesario conocer el número de soluciones M del problema de búsqueda. Sin embargo, esta es una condición que rara vez se da. Por otra parte, también es de gran utilidad saber si existe *alguna* solución al problema. Es para esto que utilizamos el algoritmo de conteo cuántico, que nos permite saber el número de soluciones del problema de búsqueda.

Supongamos que tenemos $|\alpha\rangle$ y $|\beta\rangle$ las superposiciones de no soluciones y soluciones, respectivamente, de un problema de búsqueda determinado. Si tomamos el plano formado por estos vectores, entonces podemos expresar el operador G como una rotación en este plano, utilizando como ángulo el valor θ definido en el apartado anterior, es decir, el valor θ que cumple $\cos\frac{\theta}{2} = \sqrt{\frac{N-M}{N}}$ y $\sin\frac{\theta}{2} = \sqrt{\frac{M}{N}}$. La matriz de rotación que representa G puede expresarse como:

$$G = \begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix},$$

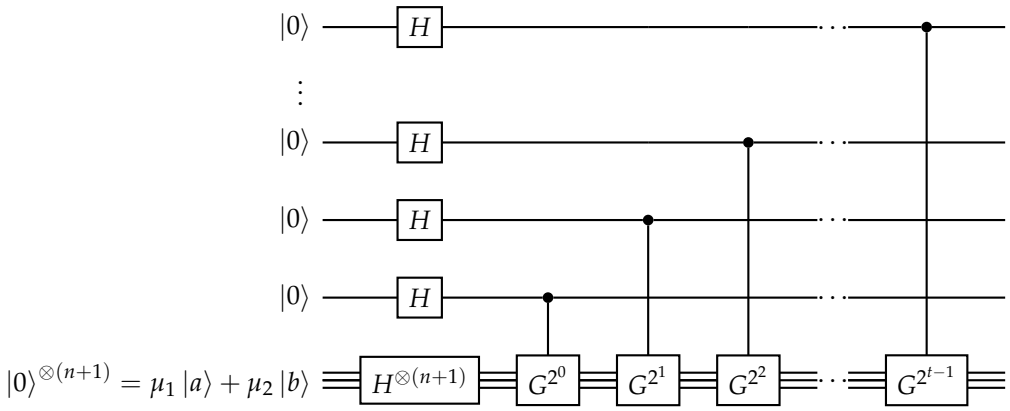
que sabemos que tiene 2 autovectores, llamémosles $|a\rangle$ y $|b\rangle$. Podemos calcular los autovalores de estos autovectores haciendo uso de su polinomio característico:

$$p(\lambda) = det(G - \lambda I) = det \begin{bmatrix} \cos\theta - \lambda & -\sin\theta \\ \sin\theta & \cos\theta - \lambda \end{bmatrix}$$
$$= \cos^2\theta - 2\cos\theta\lambda + \lambda^2 + \sin^2\theta = -2\cos\theta\lambda + \lambda^2 + 1.$$

Si resolvemos para λ, obtenemos:

$$\lambda = \cos\theta \pm \sqrt{\cos^2\theta - 1} = \cos\theta \pm i\sin\theta = e^{\pm i\theta}.$$

Ahora, si utilizamos el algoritmo de estimación de fase, usando como entrada G, podemos estimar la fase de estos autovalores, lo que nos proveería M. Primero, establecemos que como los autovectores producen una base del plano formado por $|\alpha\rangle$ y $|\beta\rangle$, entonces la superposición de todos los estados de la base computacional, $|\psi\rangle$, puede expresarse como $|\psi\rangle = \mu_1 |a\rangle + \mu_2 |b\rangle$ con $\mu_i \in \mathbb{C}$. Por lo tanto, podemos usar la estimación de fase con el circuito:



Este circuito es una versión levemente modificada del que aparece en el algoritmo de estimación de fase, pero funciona de la misma manera. Si usamos $t = m + \log\left(2 + \frac{1}{2\epsilon}\right)$ qubits en el primer registro, obtendremos una estimación $\tilde{\theta}$ de la fase θ que es exacta hasta m bits, con probabilidad $1 - \epsilon$. A partir de esta estimación, podemos obtener M, usando la fórmula original de θ: $\sin\frac{\theta}{2} = \sqrt{\frac{M}{N}}$. Además, a partir de esta formula, podemos calcular el error producido en el cálculo de M a partir del error de la estimación $\tilde{\theta}$. Si definimos $\Delta\theta = \theta - \tilde{\theta}$, entonces $|\Delta\theta| < 2^{-m}$. De esta forma, podemos definir el error en M como:

$$\frac{|\Delta M|}{N} = \left|\sin^2\frac{\theta + \Delta\theta}{2} - \sin^2\frac{\theta}{2}\right| = \left|\sin\frac{\theta + \Delta\theta}{2} + \sin\frac{\theta}{2}\right|\left|\sin\frac{\theta + \Delta\theta}{2} - \sin\frac{\theta}{2}\right|.$$

Ahora, podemos aplicar las desigualdades:

$$\left|\sin\frac{\theta+\Delta\theta}{2}-\sin\frac{\theta}{2}\right|\leq\frac{\Delta\theta}{2},$$
$$\left|\sin\frac{\theta+\Delta\theta}{2}\right|\leq\sin\frac{\theta}{2}+\frac{|\Delta\theta|}{2}.$$

Obteniendo la expresión:

$$\frac{|\Delta M|}{2N}<\left(2\sin\frac{\theta}{2}+\frac{|\Delta\theta|}{2}\right)\frac{|\Delta\theta|}{2}.$$

En ella, sustituimos $\sin\frac{\theta}{2}=\frac{M}{N}$ y $|\Delta\theta|<2^{-m}$, llegando a una cota del error de M:

$$|\Delta M|<\left(\sqrt{2MN}+\frac{N}{2^{m+1}}\right)2^{-m}.$$

5 Implementación de algoritmos cuánticos en Qiskit

Qiskit es un conjunto de herramientas de software de código abierto desarrollado por IBM que permite trabajar con computadoras cuánticas. Es una biblioteca de Python que proporciona una interfaz para simular algoritmos cuánticos en computadoras clásicas.

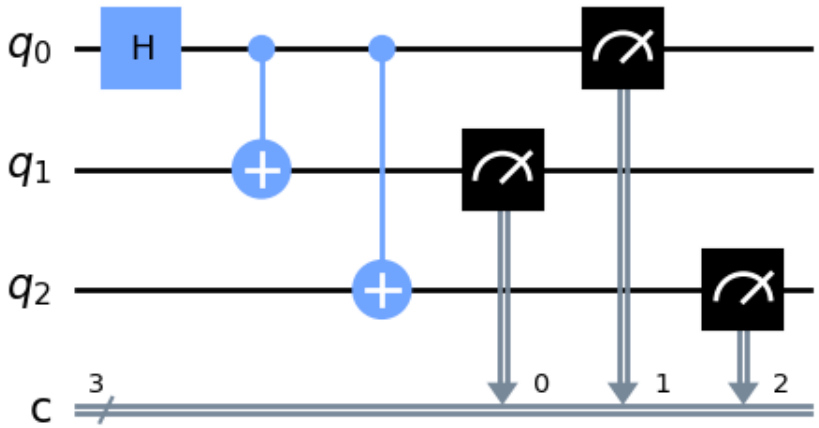
En este capítulo presentaremos un breve tutorial de como usar esta herramienta, así como la implementación de los algoritmos expuestos en el capítulo 4. Todo el código que enseñamos a continuación ha sido ejecutado en libretas Jupyter. El código también se encuentra disponible en el repositorio de Github: https://github.com/danielgonalfert/Qiskit_impl.

5.1. Introducción

[27]:
```
from qiskit import QuantumCircuit, QuantumRegister, assemble, Aer
from qiskit.quantum_info import Statevector
from qiskit_aer import AerSimulator
from qiskit.quantum_info.operators import Operator, Pauli
```

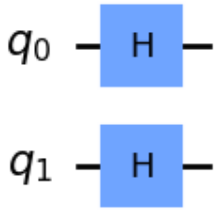
[18]:
```
#En primer lugar creamos un circuito con n qubits, con m bits clásicos. Para␣
 ↪ello, utilizamos el comando:
n=3
circ = QuantumCircuit(n,n)

#Para añadir puertas solo tenemos que escribir la puerta que queramos añadir␣
 ↪junto
#con el índice/s en los que la queramos añadir. Por ejemplo:
circ.h(0)
circ.cx(0,1)
circ.cx(0,2)

#Además, podemos añadir medidas en la base computacional. Con el siguiente␣
 ↪comando, que acepta dos vectores, generamos una
#asociación entre los índices de qubits y bits, de manera que la medida del␣
 ↪qubits [a_i] se guarda en el bits [b_i].
circ.measure([1,0,2], [0,1,2])


#Podemos dibujar el circuito con el comando:
circ.draw('mpl')
```

[18]:



[19]:
```
#También podemos crear un circuito sin medidas.
s=2
qc = QuantumCircuit(s,name = "U")
qc.h(0)
qc.h(1)

qc.draw('mpl')
```

[19]:



[20]:
```
#Podemos generar un estado y pasarlo por el circuito, con el siguiente comando,␣
 ↪que genera el estado |natural> en
#la base computacional de número de qubits tam. Por ejemplo el estado |2> en la␣
 ↪base computacional de 2 qubits:
tam = 2
natural = 0
state = Statevector.from_int(natural, 2**tam)
state.draw('latex')
```

[20]:

$$|00\rangle$$

[21]:
```
#Ahora podemos usar este estado como entrada:
salida = state.evolve(qc)
#Este comando nos devuelve una entrada poco legible.
salida
```

```
Statevector([0.5+0.j, 0.5+0.j, 0.5+0.j, 0.5+0.j],
            dims=(2, 2))
```

[22]:
```
#Podemos utilizar este para obtener una entrada mas comprensible.
salida.draw('latex')
```

[22]:

$$\frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle$$

[23]:
```
#Por otra parte, para poder utilizar la medición, tenemos que utilizar un␣
↪"backend" para compilar el circuito y convertirlo
#en un programa. Primero definimos el backend:
simulator = AerSimulator()


#Ahora compilamos el circuito:
circuit_compiled = transpile(circ,simulator)


#El resultado es un programa que podemos ejecutar.
#Corremos el programa en el simulador un número "shots" de veces.
#Al utilizar el comando run, obtenemos un objeto que trabaja asíncronamente. Con␣
↪el comando status(), obtenemos el estado
#actual, mientras que con result() obtenemos el resultado final, si es que lo␣
↪hay.
trabajo_sim = simulator.run(circuit_compiled, shots = 1024)
result = trabajo_sim.result()

result
```

[23]:
```
Result(backend_name='aer_simulator', backend_version='0.12.0', qobj_id='',
job_id='7b5faf40-990e-4eb7-a8ce-88a9ca3e64aa', success=True,
results=[ExperimentResult(shots=1024, success=True, meas_level=2,
data=ExperimentResultData(counts={'0x0': 498, '0x7': 526}),
header=QobjExperimentHeader(creg_sizes=[['c', 3]], global_phase=0.0,
memory_slots=3, metadata=None, n_qubits=3, name='circuit-124', qreg_sizes=[['q',
3]]), status=DONE, seed_simulator=2538971145,
metadata={'batched_shots_optimization': False, 'method': 'stabilizer',
'active_input_qubits': [0, 1, 2], 'device': 'CPU', 'remapped_qubits': False,
'num_qubits': 3, 'num_clbits': 3, 'sample_measure_time': 0.0082601,
'input_qubit_map': [[0, 0], [1, 1], [2, 2]], 'measure_sampling': True, 'noise':
'ideal', 'parallel_shots': 1, 'parallel_state_update': 8, 'fusion': {'enabled':
False}}, time_taken=0.033962)], date=2023-06-19T19:36:32.233637,
status=COMPLETED, header=None, metadata={'parallel_experiments': 1,
'omp_enabled': True, 'max_memory_mb': 8157, 'max_gpu_memory_mb': 0,
'num_processes_per_experiments': 1, 'mpi_rank': 0, 'num_mpi_processes': 1,
'time_taken_execute': 0.0339987}, time_taken=0.03977823257446289)
```

[24]:
```
#Como podemos ver, esta salida es bastante confusa. Podemos obtener el resultado␣
↪del experimento con el comando:
counts = result.get_counts(circuit_compiled)
print(counts)
```

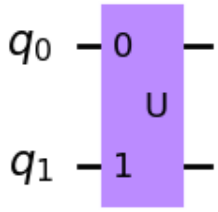
```
{'000': 498, '111': 526}
```

[28]:
```
#Para poder modularizar el proceso de desarrollo de un circuito cuántico␣
↪complejo, podemos convertir los circuitos que
#queremos en operadores de caja negra.

U = Operator(qc)
print(U)
```

```
Operator([[ 0.5+0.j,  0.5+0.j,  0.5+0.j,  0.5+0.j],
          [ 0.5+0.j, -0.5+0.j,  0.5+0.j, -0.5+0.j],
          [ 0.5+0.j,  0.5+0.j, -0.5+0.j, -0.5+0.j],
          [ 0.5+0.j, -0.5+0.j, -0.5+0.j,  0.5+0.j]],
         input_dims=(2, 2), output_dims=(2, 2))
```

[29]:
```
#Podemos añadir este operador a un circuito en sí mismo.
c = QuantumCircuit(2)
c.append(qc.to_gate(),[0,1])

c.draw('mpl')
```

[29]:



[30]:
```
#Podemos comprobar fácilmente que este operador nos da el mismo resultado que el␣
↪circuito del que proviene
state = Statevector.from_int(2, 2**2)
state.draw('latex')
salida = state.evolve(c)
salida.draw('latex')
```

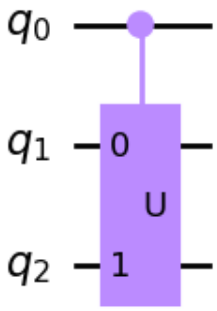
[30]:

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

[31]:
```
#También podemos añadir condicionalidad a estos operadores personalizados:
cond_c = QuantumCircuit(3)
condU = qc.to_gate().control(1)

cond_c.append(condU,[0,1,2])
```

```
cond_c.draw('mpl')
```

[31]:



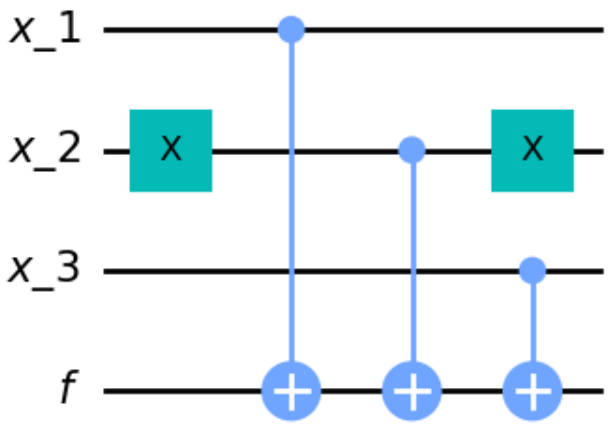
5.2. Algoritmo de Deutsch-Jozsa

[136]:
```
from qiskit import QuantumCircuit, QuantumRegister, assemble, Aer
from qiskit.quantum_info import Statevector
from qiskit_aer import AerSimulator
from qiskit.quantum_info.operators import Operator, Pauli
```

Supongamos que la función oráculo que nos da |x>|y> -> |x>|y XOR f(x) se implementa con el siguiente circuito:

[123]:
```
x_1 = QuantumRegister(1,'x_1')
x_2 = QuantumRegister(1,'x_2')
x_3 = QuantumRegister(1,'x_3')
f = QuantumRegister(1,'f')
circ = QuantumCircuit(x_1,x_2,x_3,f,name="f")

circ.x(x_2[0])
circ.cx(x_1[0],f[0])
circ.cx(x_2[0],f[0])
circ.cx(x_3[0],f[0])
circ.x(x_2[0])
circ.draw('mpl')
```

[123]:



La tabla de verdad de f es:

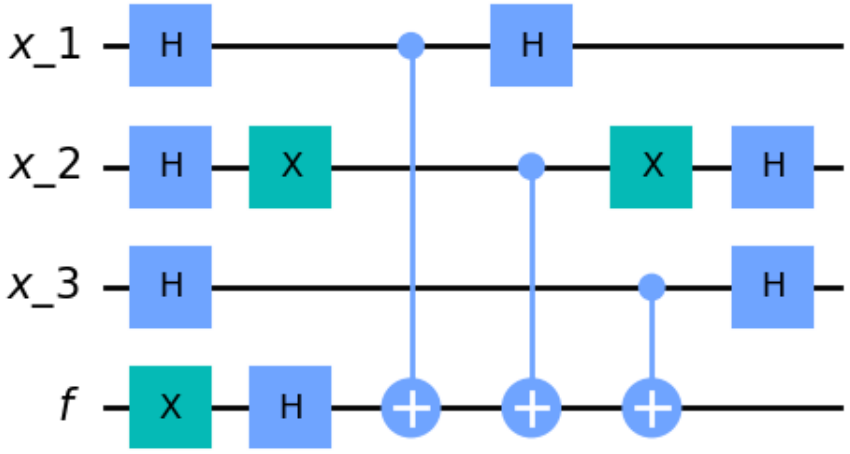
x_1	x_2	x_3	y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0

x_1	x_2	x_3	y
1	0	1	1
1	1	0	1
1	1	1	0

que claramente es balanceada. Ahora tenemos que añadir puertas de Hadamard a cada qubit antes y despues del oráculo.

[134]:
```
x_1 = QuantumRegister(1,'x_1')
x_2 = QuantumRegister(1,'x_2')
x_3 = QuantumRegister(1,'x_3')
f = QuantumRegister(1,'f')
circ = QuantumCircuit(x_1,x_2,x_3,f,name="f")

circ.h(x_1[0])
circ.h(x_2[0])
circ.h(x_3[0])
circ.x(f[0])
circ.h(f[0])



circ.x(x_2[0])
circ.cx(x_1[0],f[0])
circ.cx(x_2[0],f[0])
circ.cx(x_3[0],f[0])
circ.x(x_2[0])


circ.h(x_1[0])
circ.h(x_2[0])
circ.h(x_3[0])


circ.draw('mpl')
```

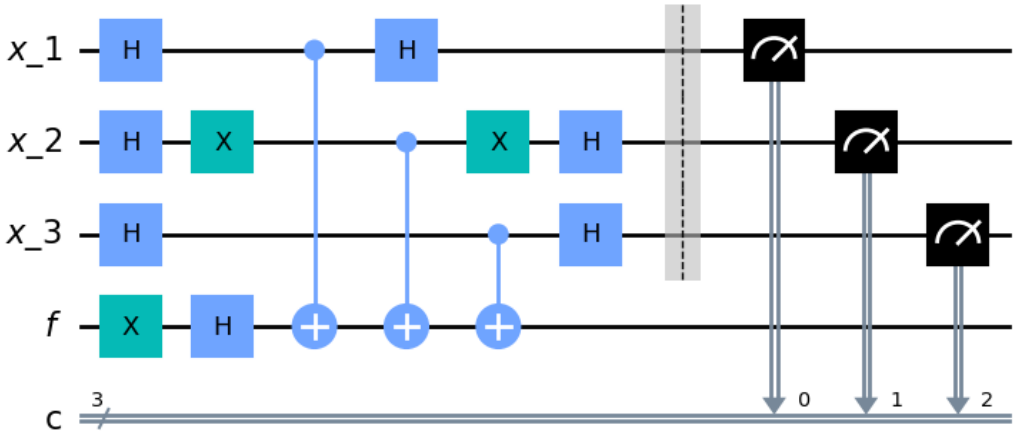
[134]:



Ahora componemos con un circuito formado por bits clásicos para añadir la medición al final.

[135]:
```
meas = QuantumCircuit(3, 3)
meas.barrier(range(3))
meas.measure(range(3), range(3))
circ.add_register(meas.cregs[0])
circ = circ.compose(meas)


circ.draw('mpl')
```

[135]:



Por último realizamos la medición.

[126]:
```
simulator = AerSimulator()
circuit_compiled = transpile(circ,simulator)
trabajo_sim = simulator.run(circuit_compiled, shots = 1024)
result = trabajo_sim.result()

counts = result.get_counts(circuit_compiled)
print(counts)
```

```
{'111': 1024}
```

Como vemos, la probabilidad de obtener el valor 000 es nula, por lo que sabemos que la función es balanceada.

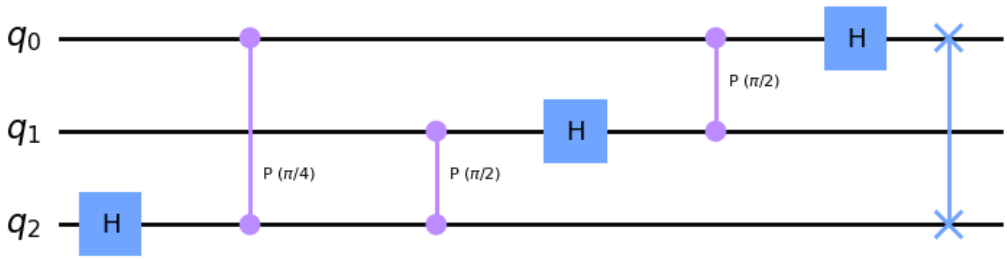
5.3. Transformada cuántica de Fourier

[3]:
```
import math
from qiskit import *
from qiskit.quantum_info.operators import Operator, Pauli
from qiskit import QuantumCircuit, QuantumRegister, assemble, Aer
from qiskit.quantum_info import Statevector
from qiskit.circuit.library import QFT, CU1Gate

from qiskit import QuantumCircuit
from qiskit.circuit import Parameter
from qiskit.transpiler import PassManager
from qiskit.transpiler.passes import Unroller
```

[11]:
```
#Ahora vamos con la función que genera el circuito

def TCF_recursivo(circuit, n):
    if n == 0:
        for i in range(int(circuit.num_qubits/2)):
            circuit.swap(i, n-i-1)
        return circuit
    n -= 1
    circuit.h(n)
    for qubit in range(n):
        circuit.cp(math.pi/2**(n-qubit), qubit, n)
    TCF_recursivo(circuit, n)

def TCF(n):
    qc = QuantumCircuit(n)
    TCF_recursivo(qc,n)
    return qc
```

[9]:
```
TCF(3).draw('mpl')
```

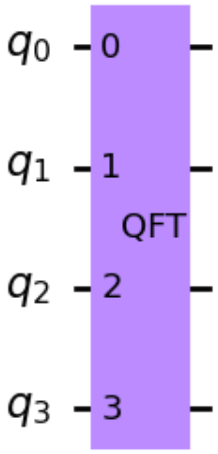
[9]:



Por otra parte, Qiskit tiene una implementación propia de este circuito, que es la que utilizaremos a partir de ahora.

[5]:
```
qc = QFT(num_qubits=4, approximation_degree=0, do_swaps=True, inverse=False,␣
↪insert_barriers=False, name=None)
qc.draw('mpl')
```

[5]:



5.4. Algoritmo de estimación de fase

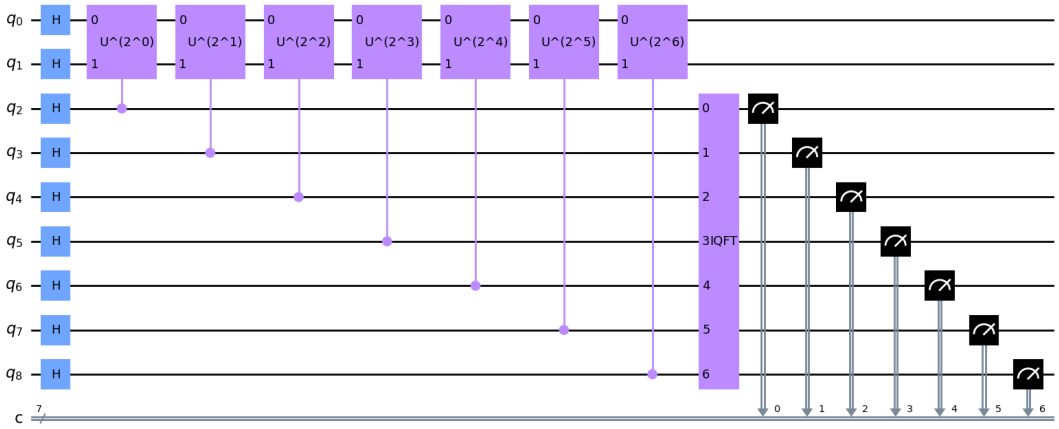
[34]:
```
import math
from qiskit.quantum_info.operators import Operator, Pauli
from qiskit import QuantumCircuit, QuantumRegister, assemble, Aer
from qiskit.quantum_info import Statevector
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
from qiskit.circuit.library import QFT, PhaseEstimation
```

Supongamos que queremos estimar las fases de los autovalores de la matriz U que definimos más abajo. Queremos medir con una precisión de n bits y probabilidad 1-epsilon, por lo que definimos t = n + E(log(2+1/(2epsilon)))

[32]:
```
eps = 0.05
n = 4

t = n + math.trunc(math.log(2+1/(2*eps)))
if math.log(2+1/(2*eps)) - math.trunc(math.log(2+1/(2*eps))) != 0:
    t = t+1

#Definimos U como el operador rotación theta
U = [[1,0,0,0],[0,1j,0,0],[0,0,-1,0],[0,0,0,-1j]]

m = int(math.log2(len(U)))
```

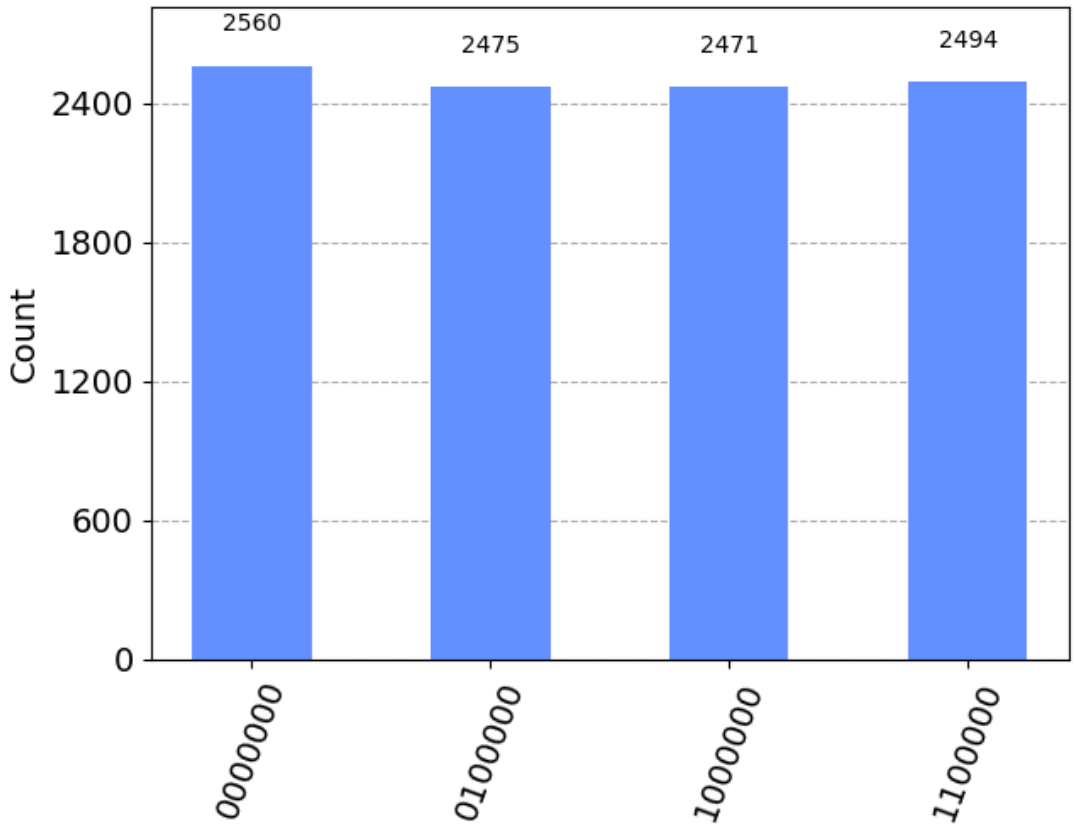
[33]:
```
phase_circ = QuantumCircuit(t+m,t)

for i in range(t+m):
    phase_circ.h(i)


for i in range(t):
    cond_operator_U = exponenciar(U,2**i)
    partial_circ = QuantumCircuit(m, name="U^(2^"+str(i)+")")
    partial_circ.append(Operator(cond_operator_U),[i for i in range(m)])
    phase_circ.append(partial_circ.to_gate().control(1),[m+i]+[i for i in␣
 ↪range(m)])

phase_circ.append(QFT(num_qubits=t, approximation_degree=0, do_swaps=True,␣
 ↪inverse=True, insert_barriers=False, name=None),[i for i in range(m,t+m)])

for i in range(m,m+t):
    phase_circ.measure(i,i-m)

phase_circ.draw('mpl')
```

[33]:



[18]:
```
#Procedemos a compilar el circuito y ejecutarlo:
simulator = AerSimulator()
circuit_compiled = transpile(phase_circ,simulator)
trabajo_sim = simulator.run(circuit_compiled, shots = 10000)
result = trabajo_sim.result()
counts = result.get_counts(circuit_compiled)

plot_histogram(counts)
```

[18]:



Como hemos discutido en el estudio teórico de este algoritmo, al medir obtenemos con las dos fases posibles de los autovectores de este operador, que son 0.000000 (eˆ(0)=1),0.100000 (eˆ(iPI)= -1),0.010000 = 1/4 (eˆ(iPI/2) = i) y 0.110000 (eˆ(iPI3/2) = -i).

5.5. Algoritmo de cálculo del orden de un natural

[23]:
```
import math
from qiskit.quantum_info.operators import Operator, Pauli
from qiskit import QuantumCircuit, QuantumRegister, assemble, Aer
from qiskit.quantum_info import Statevector
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
from qiskit.circuit.library import QFT, PhaseEstimation
from IPython.display import display,Latex
```

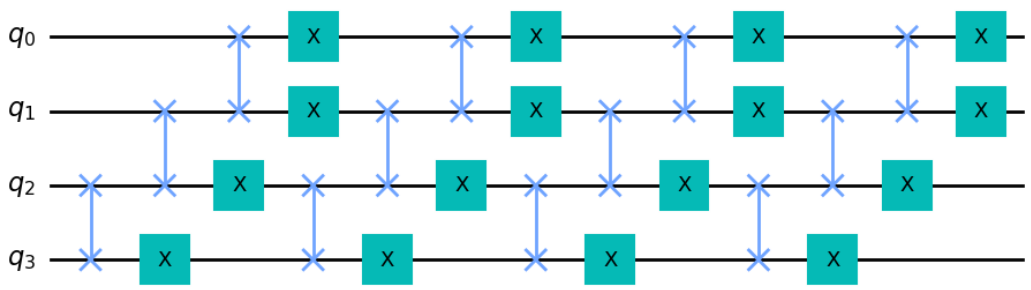
Como ya vimos en el capítulo anterior, la implementación de este algoritmo depende de poder implementar el operador:

U|y> = |xy (mod N)>.

Supongamos que queremos calcular el orden de 7 mod 15. Sabemos de antemano que 7^4 mod 15 = 1, de manera que el orden es 4. 15 se puede escribir con 4 bits. De manera que empezamos por definir que el circuito que realiza la operación x -> x7^j mod 15 y las variables asociadas al circuito.

[14]:
```
def calcular_operador_elevado_a_potencia(j):
    U_alt = QuantumCircuit(4,name="U^(2^"+str(j)+")")
    for i in range(2**j):
        U_alt.swap(2,3)
        U_alt.swap(1,2)
        U_alt.swap(0,1)
        for k in range(4):
            U_alt.x(k)
    return U_alt
calcular_operador_elevado_a_potencia(2).draw('mpl')
```

[14]:



[18]:
```
N = 15
eps = 0.05
x = 7

truncated_value = math.trunc(math.log2(N))
exact_value_log = math.log2(N)
```

```
L = truncated_value
if (truncated_value - exact_value_log != 0):
    L = L + 1

truncated_value = 2*L + math.trunc(math.log(2+1/(2*eps)))
exact_value_log = 2*L + math.log(2+1/(2*eps))

t = L + int( math.trunc(math.log(2+1/(2*eps)))) -1

if (truncated_value - exact_value_log != 0):
    t = t + 1

print(t,L)
```

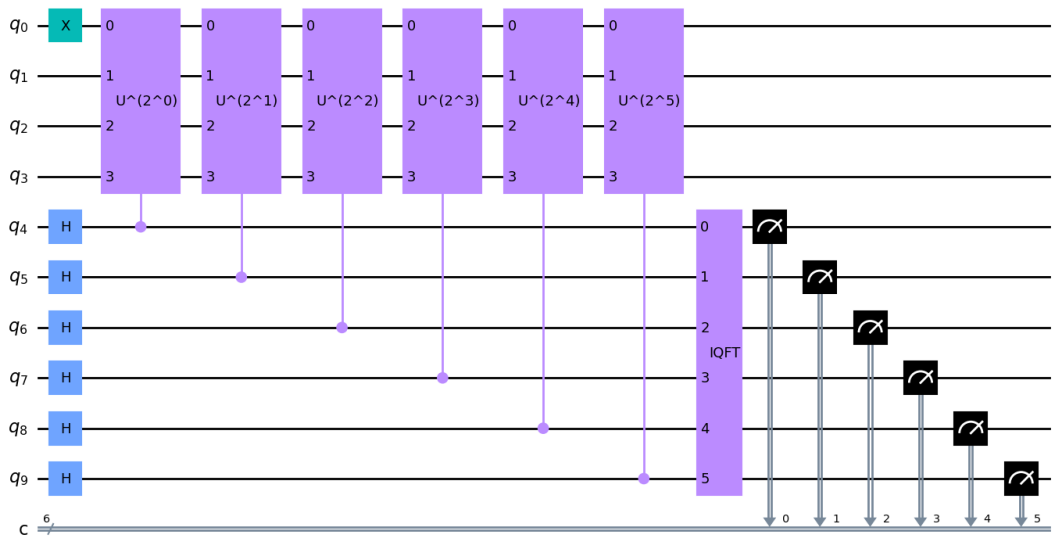
```
6 4
```

[19]:
```
def circuito_estimacion_fase(t,L,N,x):
    phase_circ = QuantumCircuit(t+L,t)


    phase_circ.x(0)
    for i in range(L,t+L):
        phase_circ.h(i)

    for i in range(t):
        #cond_operator_U = exponenciar(U,2**i)
        #partial_circ = QuantumCircuit(L, name="U^(2^"+str(i)+")")
        #partial_circ.append(Operator(cond_operator_U),[i for i in range(L)])
        phase_circ.append(calcular_operador_elevado_a_potencia(i).to_gate().
↪control(1),[L+i]+[i for i in range(L)])

    phase_circ.append(QFT(num_qubits=t, approximation_degree=0, do_swaps=True,␣
↪inverse=True, insert_barriers=False, name=None),[i for i in range(L,t+L)])

    for i in range(t):
        phase_circ.measure(L+i,i)

    return phase_circ
```

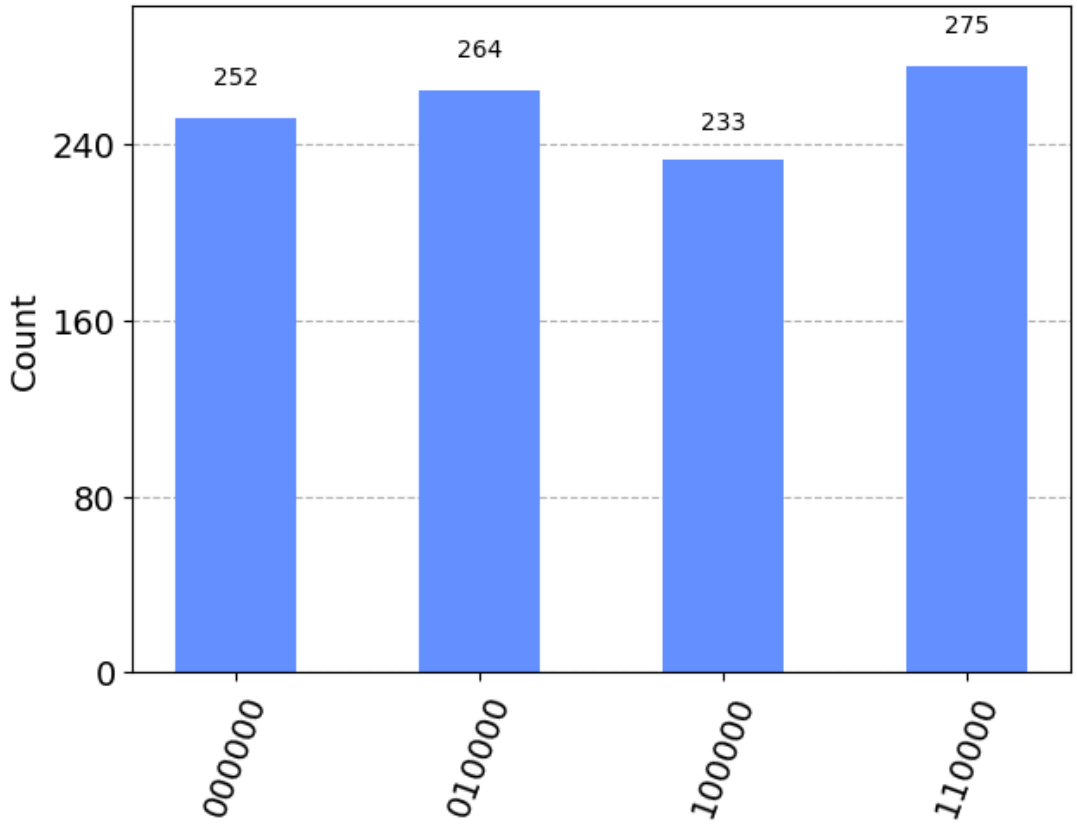
Ahora solo tenemos que aplicar el algoritmo de estimación de fase con el operador U calculado anteriormente

[20]:
```
circ = circuito_estimacion_fase(t,L,N,x)
circ.draw('mpl')
```

[20]:



[21]:
```
#Procedemos a compilar el circuito y ejecutarlo:
simulator = AerSimulator()
circuit_compiled = transpile(circ,simulator)
trabajo_sim = simulator.run(circuit_compiled, shots = 1024)
result = trabajo_sim.result()
counts = result.get_counts(circuit_compiled)

plot_histogram(counts)
```
[21]:



Aquí podemos ver como los resultados son 0,1/2,3/4 y 1/4. Como $1/2 = 2/4$ es fácil concluir que el orden buscado es 4, que es justamente lo que habíamos calculado previamente.

5.6. Algoritmo de Shor

Vamos a calcular uno de los factores del número 15.

[4]:
```
import math
from qiskit.quantum_info.operators import Operator, Pauli
from qiskit import QuantumCircuit, QuantumRegister, assemble, Aer
from qiskit.quantum_info import Statevector
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
import random
```

1.- Vemos si 15 es par.

[3]:
```
15 % 2
```

[3]:
```
1
```

2.- Vemos si existen a y b tales que aˆb = 15.

[8]:
```
L = 4
resultado = False

for b in range(1,L):
    x = L/b
    exponencial = 2**x
    u_1 = math.trunc(2**x)
    u_2 = u_1 + 1
    if u_1**b == N or u_1**b == N:
        resultado = True

resultado
```

[8]:
```
False
```

3.- Elegimos un número aleatorio entre 1 y 14. Elijamos por conveniencia el 7. mcd(7,15) = 1, de modos que pasamos al siguiente paso.

4.- Calculámos el orden de 7 mod 15. Ya hemos calculado en el apartado anterior que es 4.

5.- Como r=4 es par, podemos calcular xˆ(r/2) = 7ˆ2 = 59 != -1 mod 15, por lo tanto calculamos mcd(48,15) y mcd(50,15)

[14]:
```
def mcd(num1,num2):
    while num2 != 0:
            temp = num2
            num2 = num1 % num2
            num1 = temp
    return num1

[mcd(48,15),mcd(50,15)]
```

```
[14]: [3, 5]
```

Que como vemos son dos dos factores distintos del número 15.

5.7. Algoritmos basados en la iteración de Grover

[82]:
```
import math
from qiskit.quantum_info.operators import Operator, Pauli
from qiskit import QuantumCircuit, QuantumRegister, assemble, Aer
from qiskit.quantum_info import Statevector
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
from qiskit.circuit.library import QFT, PhaseEstimation
```

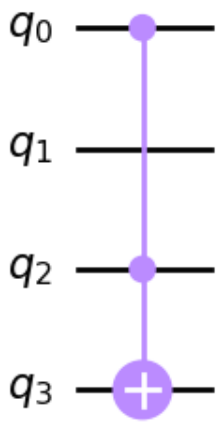
Supongamos que tenemos un problema de búsqueda cuyas soluciones son los elementos que tienen imagen 1 por la función representada por la siguiente tabla de verdad:

x_1	x_2	x_3	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

A continución construimos el circuito que realiza la operación |x>|q> -> |x>|q XOR f(x)>.

[171]:
```
N = 8
M = 2


G = QuantumCircuit(3+1, name="G")

G.toffoli(0,2,3)



G.draw('mpl')
```

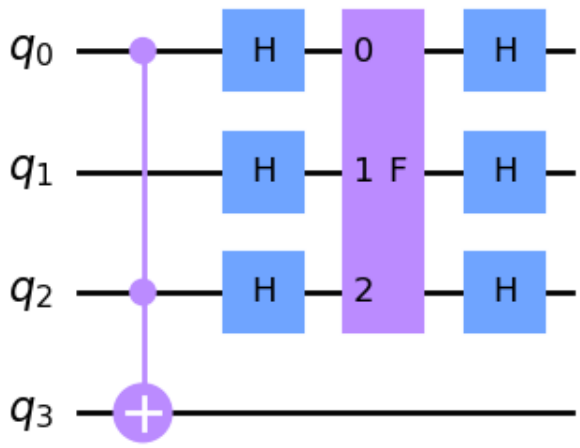
[171]:



Ahora construimos la puerta que multiplica por -1 la amplitud de todos los estados distintos de |000>

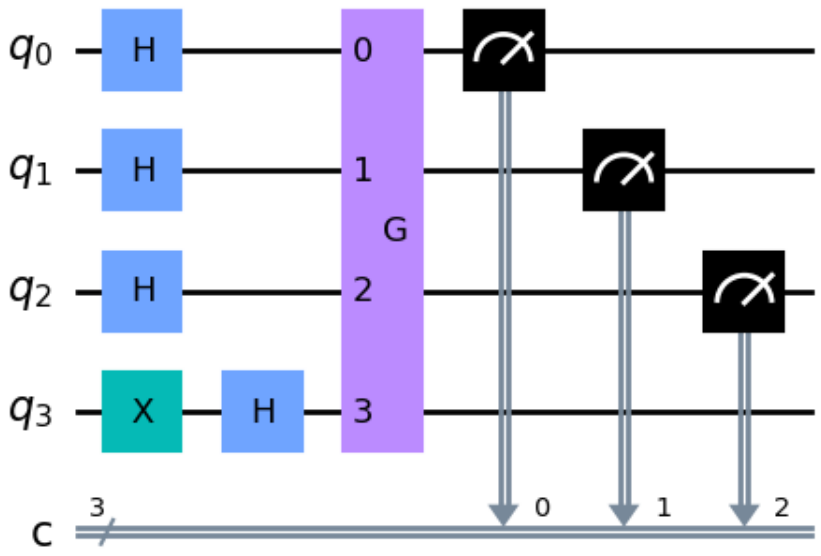
[172]:
```
qc_fase = QuantumCircuit(3,name = "F")

p = []
for i in range(2**3):
    fila = [0 for j in range(2**3)]
    fila[i] = -1
    p.append(fila)

p[0][0] = 1

G.h(0)
G.h(1)
G.h(2)


qc_fase.append(Operator(p),[0,1,2])

G.append(qc_fase.to_gate(),[0,1,2])

G.h(0)
G.h(1)
G.h(2)

G.draw('mpl')
```

[172]:



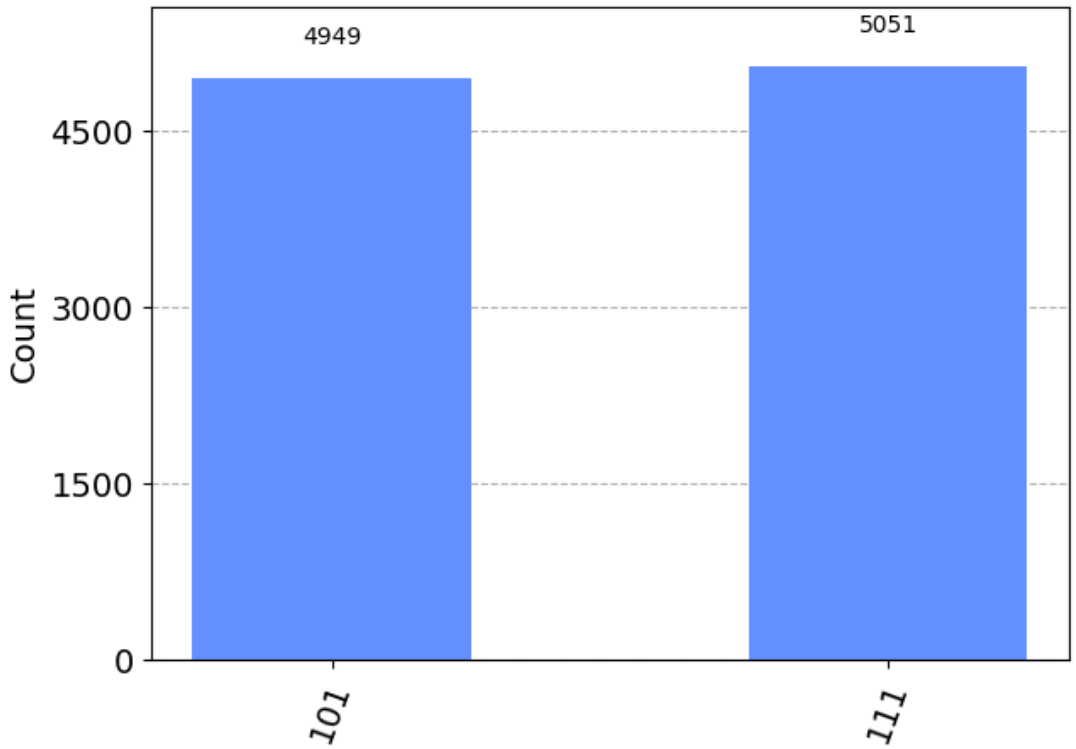
Este circuito conformará el operador de Grover G.

[85]:
```
final_circ = QuantumCircuit(4,3)
final_circ.h(0)
final_circ.h(1)
final_circ.h(2)
final_circ.x(3)
final_circ.h(3)

R = int(math.pi/4*math.sqrt(N/M))

for i in range(R):
    final_circ.append(G.to_gate(),[0,1,2,3])

for i in range(3):
    final_circ.measure(i,i)

final_circ.draw('mpl')
```
[85]:



[86]:
```
simulator = AerSimulator()
circuit_compiled = transpile(final_circ,simulator)
trabajo_sim = simulator.run(circuit_compiled, shots = 10000)
result = trabajo_sim.result()
counts = result.get_counts(circuit_compiled)

plot_histogram(counts)
```

[86]:



Como podemos ver, el resultado son las entradas tal que la función tiene imagen uno y por tanto son soluciones del problema de búsqueda.

A continución usaremos el algoritmo de conteo para estimar M (número de soluciones).

[173]:
```
eps = 0.05 #Probabilidad de error deseada.
n = 4 # Número de bits de precisión.

m = 4 # Numero de qubits necesarios para la implementación del operador de␣
 ↪Grover.

t = n + math.trunc(math.log(2+1/(2*eps)))
if math.log(n+1/(2*eps)) - math.trunc(math.log(n+1/(2*eps))) != 0:
    t = t+1

t #Número de qubits de trabajo
```

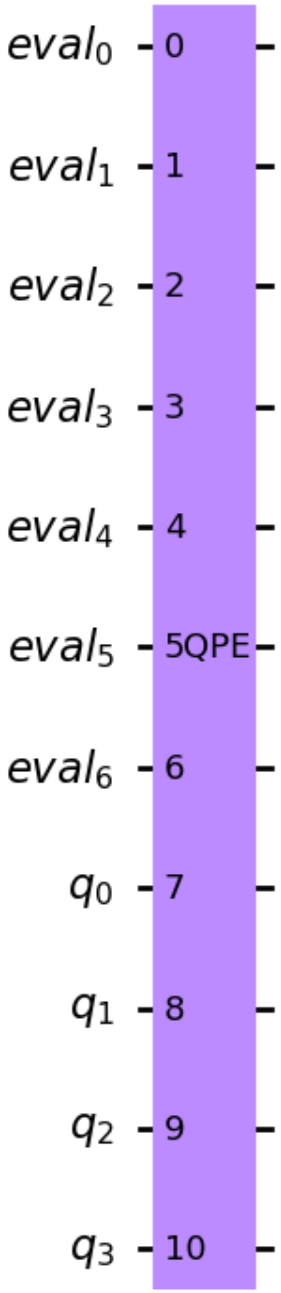
[173]:
```
7
```

Esta vez utilizaremos la implementación nativa de Qiskit del circuito de estimación de fase.

[174]:
```
#Aquí, el operador cuyas fases de autovalores queremos estimar es claramente el␣
 ↪operador de Grover, G.
```

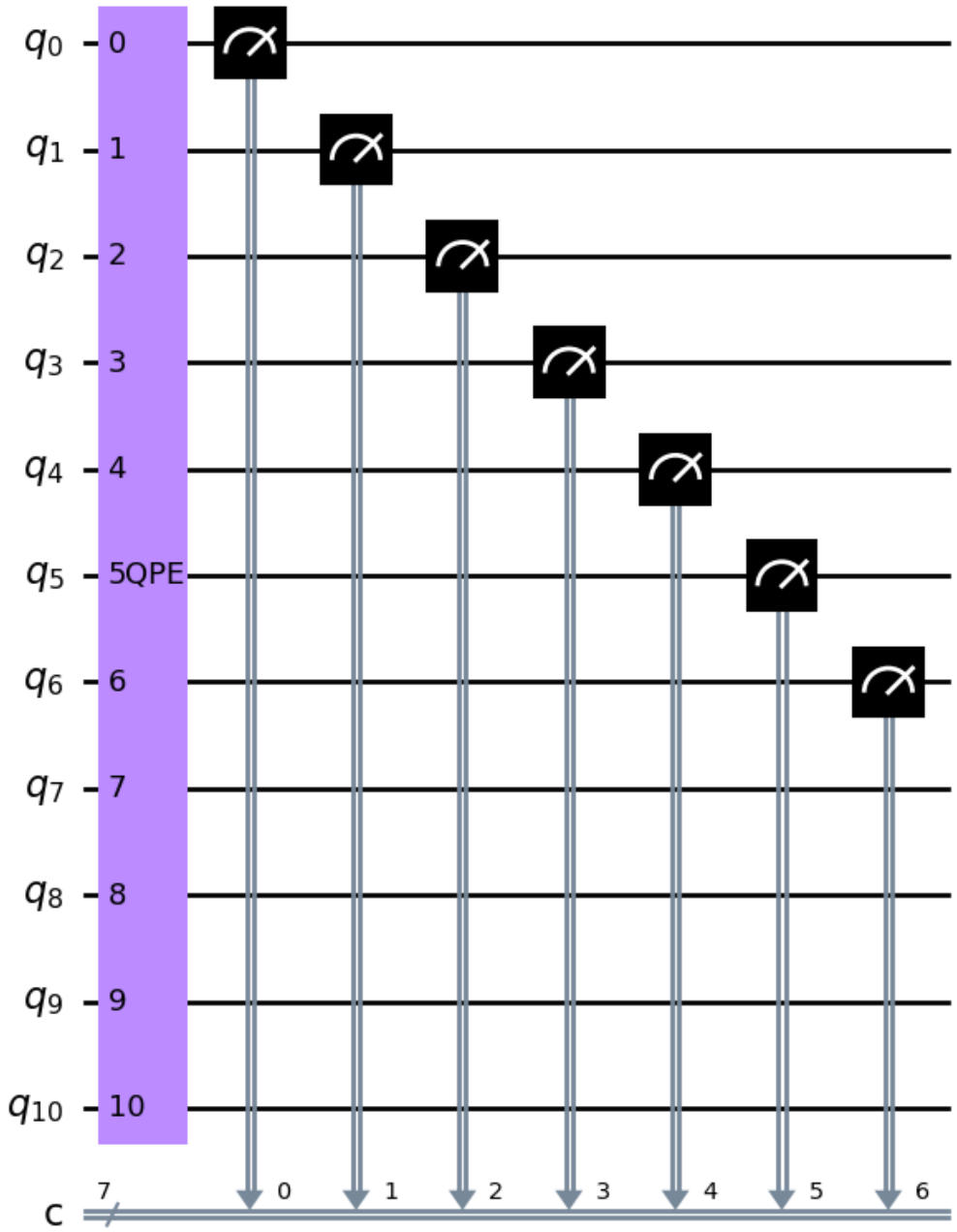
```
pc = PhaseEstimation(t,G, iqft=None, name='QPE')
pc.draw('mpl')
```

[174]:



[176]:
```
#Añadimos la medición.
pc_f = QuantumCircuit(t+m,t)
pc_f.append(pc.to_gate(),[i for i in range(t+m)])

for i in range(t):
    pc_f.measure(i,i)

pc_f.draw('mpl')
```

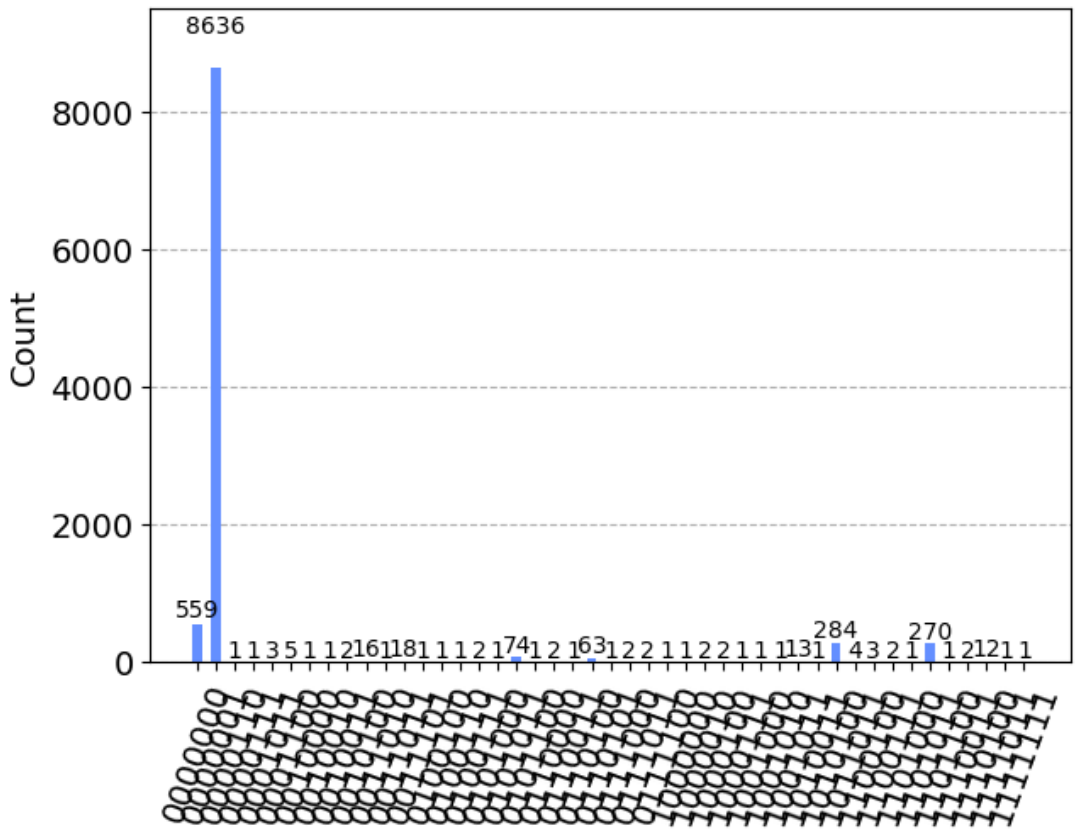
[176]:



[179]:
```
simulator = AerSimulator()
circuit_compiled = transpile(pc_f,simulator)
trabajo_sim = simulator.run(circuit_compiled, shots = 10000)
result = trabajo_sim.result()
```

```
counts = result.get_counts(circuit_compiled)

plot_histogram(counts)
```

[179]:



Ahora debemos tener en cuenta que el algoritmo de estimación de false calcula phi tal que eˆ(2iPI phi) sea un autovalor, pero necesitamos calcular theta tal que eˆ(i theta) sea dicho autovalor. Es por esto que tenemos que calcular theta a partir de phi.

[180]:
```
measured_str = max(counts, key=counts.get)
measured_int = int(measured_str)

theta = measured_int * 2*math.pi /2**t
theta
```

[180]: 0.04908738521234052

A partir de theta podemos calcular M

[169]:
```
N*(math.sin(theta/2))**2
```

[169]: 2.0

6 Conclusiones

El trabajo presentado ha realizado una exploración sistemática y exhaustiva de los fundamentos teóricos y prácticos de la computación cuántica, abarcando desde los conceptos matemáticos indispensables hasta el desarrollo teórico de algoritmos avanzados. A lo largo del trabajo se ha presentado una visión clara del potencial revolucionario de la computación cuántica en el campo de la informática.

Además de abordar los aspectos teóricos, se ha destacado la relevancia práctica de la computación cuántica mediante la implementación de algoritmos en mediante Qiskit. Esto ha permitido materializar los conceptos estudiados y evidenciar cómo las teorías y los principios explicados en la explicación teórica de los algoritmos se traducen en aplicaciones concretas.

Es importante destacar que se han logrado todos los objetivos propuestos al inicio del trabajo. Se ha logrado una comprensión sólida de los conceptos matemáticos y físicos fundamentales de la computación cuántica, así como una exploración exhaustiva de las puertas lógicas cuánticas, los circuitos cuánticos y los algoritmos cuánticos más relevantes, como el algoritmo de Shor y el algoritmo de búsqueda de Grover.

En resumen, este trabajo de fin de grado puede contribuir a brindar una base teórica sólida y una perspectiva práctica a cualquier lector poco familiarizado con los conceptos mencionados anteriormente. El trabajo realizado abre el camino para futuras investigaciones y desarrollos en el campo de la computación cuántica, que promete revolucionar la forma en que procesamos, almacenamos y transmitimos información.

Bibliografía

Las referencias se listan por orden alfabético. Aquellas referencias con más de un autor están ordenadas de acuerdo con el primer autor.

[AKS20] Anand Mohan Ashutosh Kumar Singh, Masahiro Fujita. Design and testing of reversible logic. https://link.springer.com/book/10.1007/978-981-13-8821-7, 2020. Springer Singapore.

[BAP+12] Antoine Bérut, Artak Arakelyan, Artyom Petrosyan, Sergio Ciliberto, Raoul Dillenschneider, and Eric Lutz. Experimental verification of landauer's principle linking information and thermodynamics. https://www.nature.com/articles/nature10872, Mar 2012. Nature.

[FHA98] Richard Phillips Feynman, J. G. Hey, and Robin W. Allen. Feynman lectures on computation. https://theswissbay.ch/pdf/Gentoomen%20Library/Extra/Richard_P._Feynman-Feynman_Lectures_on_Computation__-Addison-Wesley%281996%29.pdf, 1998. Addison-Wesley Longman Publishing Co., Inc.

[Fra05] Michael P. Frank. Introduction to reversible computing: Motivation, progress, and challenges. https://dl.acm.org/doi/10.1145/1062261.1062324, 2005. Association for Computing Machinery.

[MA02] Nitin Saxena Manindra Agrawal, Neeraj Kayal. Primes is in p. https://www.cse.iitk.ac.in/users/manindra/algebra/primality_v6.pdf, 2002.

[MMD03] D.M. Miller, D. Maslov, and G.W. Dueck. A transformation based algorithm for reversible logic synthesis. https://ieeexplore.ieee.org/abstract/document/1219016, 2003. IEEE Xplore.

[NC10] Michael A. Nielsen and Isaac L. Chuang. Quantum computation and quantum information: 10th anniversary edition. http://mmrc.amss.cas.cn/tlb/201702/W020170224608149940643.pdf, 2010. Cambridge University Press.

[Tah16] Saleem Taha. Reversible logic synthesis methodologies with application to quantum computing. https://link.springer.com/book/10.1007/978-3-319-23479-3, 01 2016.