UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

SIMULAÇÃO DE ALGORITMOS QUÂNTICOS CONSIDERANDO A TEORIA QUÂNTICA MODAL EM HASKELL

DISSERTAÇÃO DE MESTRADO

Eduardo Ferreira da Silva

Santa Maria, RS, Brasil

2014

SIMULAÇÃO DE ALGORITMOS QUÂNTICOS CONSIDERANDO A TEORIA QUÂNTICA MODAL EM HASKELL

Eduardo Ferreira da Silva

Dissertação apresentada ao Curso de Mestrado em Computação do Programa de Pós-Graduação em Informática da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de

Mestre em Ciência da Computação

Orientadora: Profª. Drª. Juliana Kaizer Vizzotto

Santa Maria, RS, Brasil

2014

Ferreira da Silva, Eduardo

Simulação de Algoritmos Quânticos Considerando a Teoria Quântica Modal em Haskell / por Eduardo Ferreira da Silva. – 2014.
76 f.: il.; 30 cm.

Orientadora: Juliana Kaizer Vizzotto
Dissertação (Mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Informática, RS, 2014.

1. Computação Quântica. 2. Algoritmos Quânticos. 3. Teoria Modal Quântica. I. Vizzotto, Juliana Kaizer. II. Título.

E-mail: eduardo.ferreira1983@gmail.com

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

SIMULAÇÃO DE ALGORITMOS QUÂNTICOS CONSIDERANDO A TEORIA QUÂNTICA MODAL EM HASKELL

elaborada por
Eduardo Ferreira da Silva

como requisito parcial para obtenção do grau de
Mestre em Ciência da Computação

COMISSÃO EXAMINADORA:

Juliana Kaizer Vizzotto, Drª.
(Presidente/Orientadora)

André Rauber Du Bois, Dr. (UFPel)

Eduardo Kessler Piveta, Dr. (UFSM)

Santa Maria, 29 de Agosto de 2014.

AGRADECIMENTOS

Agradeço à instituição UFSM pela oportunidade de pós-graduação e também à CAPES, órgão que proporcionou a bolsa de estudos que foi fundamental no processo de formação.

Ao meu amigo Regis Schuch, pelo incentivo inicial e principal influência para de fato realizar a inscrição para o processo de seleção. Foi meu colega tanto de estudo quanto no convívio pessoal.

À Professora Juliana Kaizer Vizzotto, minha orientadora e uma pessoa exemplar, agradeço pelos valiosos conselhos, puxões de orelha, dicas e principalmente pela confiança no trabalho proposto.

À minha família, pelo suporte emocional e incentivo ao estudo que tive desde o começo. Principalmente à minha mãe Maria Adiles, pela força, afeto e estrutura sólida que ela representa.

Aos meus irmãos, que têm papel fundamental para o crescimento pessoal, ao apoio sempre prestado e por estarem presentes em todas as fases da minha vida.

À minha namorada Carline, pela compreensão, amor e tudo o que representa na minha vida, me incentivando nos momentos difíceis durante o processo de formação.

Aos meus amigos, que fazem parte da minha vida, pelo companheirismo, por proporcionar a troca de ideias e bons momentos.

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

SIMULAÇÃO DE ALGORITMOS QUÂNTICOS CONSIDERANDO A TEORIA QUÂNTICA MODAL EM HASKELL

AUTOR: EDUARDO FERREIRA DA SILVA
ORIENTADORA: JULIANA KAIZER VIZZOTTO
Local da Defesa e Data: Santa Maria, 29 de Agosto de 2014.

O desenvolvimento de algoritmos quânticos esbarra em algumas dificuldades, tais como a falta de um computador quântico de propósito geral, uma linguagem ou formalismo específico, ou o alto consumo de recursos computacionais quando trata-se de simulação de algoritmos quânticos em computadores clássicos. Este trabalho apresenta o desenvolvimento da biblioteca MQS (*Modal Quantum Simulation*), um modelo de computação quântica escrita em Haskell capaz de desenvolver a implementação de algoritmos considerando a Teoria Quântica Modal. Essa teoria tem como atrativo a substituição de amplitudes de probabilidade complexas por campos finitos. Ou seja, uma teoria simplificada se comparada com a computação quântica convencional, mas que mantém muito da estrutura da computação quântica tradicional.

Palavras-chave: Computação Quântica. Algoritmos Quânticos. Teoria Modal Quântica.

ABSTRACT

Master's Dissertation
Post-Graduate Program in Informatics
Federal University of Santa Maria

SIMULATION OF QUANTUM ALGORITHMS CONSIDERING THE QUANTUM THEORY MODAL IN HASKELL

AUTHOR: EDUARDO FERREIRA DA SILVA
ADVISOR: JULIANA KAIZER VIZZOTTO
Defense Place and Date: Santa Maria, March 29st, 2014.

The development of quantum algorithms faces some difficulties such as the lack of a general-purpose quantum computer, a language or a specific formalism, or high consumption of computational resources when it is simulation of quantum algorithms on classical computers. This paper presents the development of the MQS (Modal Quantum Simulatio), a library of quantum computing written in Haskell able to develop implementation of algorithms considering the Quantum Theory Modal. This theory is attractive as the replacement of complex probability amplitudes for finite fields. A simplified theory is compared with the conventional quantum computing, but retains much of the structure of the traditional quantum computing.

Keywords: Quantum Computing. Quantum Algorithm. Modal Quantum Theory.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

UFSM	Universidade Federal de Santa Maria
UFPEL	Universidade Federal de Pelotas
Qubit	Bit Quântico
Mobit	Bit Quântico Discreto
CQ	Computação Quântica
TQM	Teoria Quântica Modal
MQS	*Modal Quantum Simulation*

LISTA DE SÍMBOLOS

$\|\psi\rangle$	Bit Quântico
$\|\psi)$	Bit Quântico Modal
α	Alpha
β	Beta
ϕ	Phi
ψ	Psi

SUMÁRIO

1 INTRODUÇÃO

A Computação Quântica (CQ) é a área que estuda a aplicação das teorias e propriedades da física quântica na ciência da computação. O foco principal da CQ é o desenvolvimento de um computador quântico, i.e., um computador capaz de realizar o processamento de informações através de um sistema quântico, baseado em propriedades atômicas (NIELSEN; CHUANG, 2000).

Os primeiros trabalhos sobre computação quântica foram introduzidos por Richard Feynmann (FEYNMAN; SHOR, 1982) e David Deutsch (DEUTSCH, 1985) até a metade da década de 1980. Sendo que o primeiro trouxe uma visão inicial de computador baseado em efeitos quânticos, enquanto o segundo provou através da máquina de Turing quântica que um computador clássico não poderia simular em tempo polinomial um computador quântico.

Pode-se creditar a Peter Shor (SHOR, 1994) um dos primeiros avanços mais significativos na área de computação quântica. Shor desenvolveu um algoritmo quântico conhecido como *Algoritmo de Shor*, baseado em um computador quântico hipotético capaz de resolver eficientemente a fatoração de números inteiros grandes em tempo polinomial. Atualmente, um dos algoritmos mais seguros utilizados na criptografia da informação é o algoritmo RSA[1], ele envolve chaves públicas e privadas baseadas justamente na fatoração de números inteiros grandes. Por esta razão, evidencia-se que o algoritmo de Shor tem um impacto direto no processo de criptografia de informações.

Também em 1996, Grover (1996) definiu um algoritmo quântico para pesquisa em bases de dados não estruturadas. Esse problema considera uma base com N elementos não ordenados e tem como objetivo buscar um elemento específico. O melhor algoritmo clássico resolve o problema em $O(N)$ passos. Surpreendentemente, o algoritmo de Grover resolve o problema $O(\sqrt{N})$ passos devido às propriedades da mecânica quântica. Pode-se dizer que a definição desses dois algoritmos impulsionou o desenvolvimento da área.

Por outro lado, o desenvolvimento de bons algoritmos quânticos comumente esbarra em duas dificuldades: a primeira deve-se à maneira de pensar, que está intuitivamente conectada com ideias clássicas; a segunda é que para um algoritmo ser interessante ao meio da computação, não deve-se apenas considerar elementos da mecânica quântica, ou seja, seu desenvolvimento deve estar em um nível mais próximo ao cientista da computação (NIELSEN; CHUANG,

[1] Nomeado RSA devido aos autores Ronald Rivest, Adi Shamir e Leonard Adleman.

2000).

Além disso, atualmente não existem computadores quânticos de propósito geral. Uma proposta, ainda investigativa, de computador quântico é produzido pela empresa *D-Wave Systems Inc.*, que recentemente teve um modelo adquirido pelas empresas Google e Nasa (SANZONE; SADLER, 2014)

Assim, a investigação prática de algoritmos quânticos nos dias de hoje se dá através da simulação em computadores clássicos. A simulação da computação quântica é uma maneira plausível de testar e verificar (considerando as limitações físicas) os algoritmos quânticos. Entretanto, uma barreira encontrada na simulação de algoritmos quânticos é a elevada demanda de recursos computacionais, tanto de memória, quanto de processamento.

Essa demanda é ocasionada principalmente, devido à vasta quantidade de informações que um qubit (bit quântico) carrega implicitamente devido as amplitudes complexas associadas ao estado do qubit. Além disso, existe a complexidade matemática envolvida nos modelos quânticos, que dificulta a modelagem de um sistema computacional (SABRY, 2003).

Recentemente, Schumacher e Westmoreland (2011) propuseram uma versão simplificada da mecânica quântica, chamada Teoria Quântica Modal. Essa teoria considera campos finitos como escalares para o espaço vetorial, ao invés de números complexos. Essa simplificação é considerável e colapsa muito da estrutura complexa infinita da mecânica quântica, entretanto mantém várias características interessantes como superposição, interferência e emaranhamento, possibilitando assim a modelagem natural de protocolos de informação quânticas.

Nesse contexto, o propósito deste trabalho é desenvolver uma biblioteca em Haskell nomeada de MQS (*Modal Quantum Simulation*) para simulação de algoritmos quânticos em computadores clássicos considerando a Teoria Quântica Modal.

Ressalta-se que linguagens funcionais, em especial a linguagem Haskell foi utilizada na implementação de bibliotecas e linguagens de domínio específico para computação quântica, por exemplo Vizzotto, Altenkirch e Sabry (2006), Mu e Bird (2001) e Altenkirch e Green (2011). Acredita-se que a linguagem Haskell, por ser uma linguagem funcional, possibilita a codificação do formalismo matemático considerado na computação quântica. Além disso, a programação com mônadas, segundo Moggi (1989), são utilizadas para estruturar as computações quânticas na biblioteca é possível na linguagem Haskell.

1.1 Organização do Texto

Esta dissertação está organizada da seguinte forma: o Capítulo 2 apresenta os principais conceitos da computação quântica, bem como alguns algoritmos quânticos. O objetivo desse capítulo, além de servir como fundamentação teórica para este estudo, é oferecer uma fonte de pesquisa para futuros pesquisadores interessados no assunto. O Capítulo 3 revisa questões relativas à Computação Quântica Modal, um modelo simplificado se comparado com a computação quântica tradicional, e que é o elemento central no desenvolvimento do trabalho. O Capítulo 4, refere-se à uma investigação sobre linguagens funcionais, com uma abordagem conceitual, até exemplos de construções práticas de funções e estruturas quânticas, incluindo um estudo sobre mônadas e sua utilização na computação quântica. No Capítulo 5 recapitula-se os trabalhos que utilizam um modelo monádico em Haskell para a estruturação de efeitos quânticos. O Capítulo 6 descreve a implementação do modelo Quântico Modal em Haskell, as abordagens utilizadas para a construção de campos finitos e a forma do tratamento de efeitos computacionais, além da implementação prática dos algoritmos quânticos. Finalizando, o Capítulo 7 apresenta as considerações finais e resultados.

2 COMPUTAÇÃO QUÂNTICA

O computador clássico é baseado na arquitetura de Von Neumann, que apresenta uma distinção clara entre armazenamento e processamento dos dados. Possui processador e memória destacados por um barramento de comunicação, sendo que seu processamento é sequencial. A tecnologia base dessa arquitetura é a CMOS[2], que utiliza fundamentalmente transistores para a construção da lógica digital, contadores, registradores, dentre outros elementos computacionais (FEYNMAN, 2000).

Contudo, atualmente essa tecnologia depara-se com um limite físico e a expansão dela esbarra em problemas de níveis atômicos, em outras palavras, o tamanho no canal do transistor não pode ser reduzido sem que haja interferências atômicas, i.e., como consequência pode ocorrer instabilidade na computação, e portanto resultados imprecisos (EKERT; HAYDEN; INAMORI, 2000).

A computação quântica, por sua vez, é baseada nos fenômenos e leis da Mecânica Quântica que são fundamentalmente diferentes da física clássica. Sendo as principais características da computação quântica oriundas das propriedades da Mecânica Quântica: amplitudes complexas de probabilidade, interferência e paralelismo quântico, emaranhamento, evolução quântica unitária, entre outros (NIELSEN; CHUANG, 2000).

A estrutura matemática utilizada para descrever a Mecânica Quântica é o formalismo do espaço de Hilbert. Esse é um espaço complexo de dimensões finitas ou infinitas, dotado de produto interno, ou seja, com noções de espaço e ângulos. Sendo assim, estados puros de sistemas quânticos são considerados vetores em um espaço de Hilbert e pode-se evidenciar também que cada sistema quântico isolado corresponde a um espaço de Hilbert (GRUSKA, 1999).

Sabe-se que apenas o estudo da informação e a computação quântica requerem um grande tempo de estudo. Neste capítulo realiza-se um breve estudo das propriedades da CQ necessárias para o andamento do trabalho e por conseguinte o desenvolvimento de algoritmos quânticos. Ressalta-se ainda que, este estudo está baseado em (NIELSEN; CHUANG, 2000).

O conteúdo deste capítulo está organizado da seguinte forma: a seção 2.1 apresenta os sistemas quânticos básicos para a computação quântica, ou seja, bits quânticos. A seção 2.2 destaca como são realizadas as alterações no estado de sistemas quânticos através de transfor-

[2] *Complementary Metal Oxide Semiconductor*

mações unitárias e da medida quântica. A seção 2.3 demonstra como os algoritmos quânticos são denotados através de circuitos quânticos. A seção 2.4 está destinada a exemplificar o conceito do paralelismo quântico, característica de suma importância em algoritmos quânticos. A seção 2.5 investiga alguns dos principais algoritmos quânticos. Na seção 2.6, apresenta a síntese do capítulo com as principais características e referências bibliográficas complementares referentes a computação quântica.

2.1 Bits Quânticos

A informação na computação clássica é denotada através de bits, sendo que eles podem representar os valores 0 ou 1. Analogamente, dois estados possíveis para o qubit (*bit quântico*) são os estados fundamentais $|0\rangle$ e $|1\rangle$ que correspondem aos valores 0 e 1 clássicos. A notação "$|\,\rangle$" é o padrão utilizado para os estados da Mecânica Quântica, denominada de *notação de Dirac*. A diferença entre bits e qubits é que um qubit pode estar em infinitas posições além dos estados $|0\rangle$ e $|1\rangle$, ou seja, além dos estados fundamentais é possível formar combinações lineares de estados, conhecidas como superposições:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,. \tag{2.1}$$

O estado do qubit é representado por $|\psi\rangle$, sendo α e β números complexos que representam amplitudes de probabilidade. Os estados $|0\rangle$ e $|1\rangle$ são conhecidos como estados básicos computacionais e formam uma base ortonormal para o espaço vetorial do qubit.

A capacidade de um qubit estar em superposição de estados vai contra o conceito compreendido pela física clássica. Um bit clássico, por exemplo, pode ser comparado com uma moeda. Uma vez que lança-se uma moeda ao ar, após o repouso ela assumirá um dos estados, cara ou coroa e, portanto, não existe um estado intermediário. Por outro lado, um qubit, pode estar em um estado de superposição, ou seja, contínuo entre $|0\rangle$ e $|1\rangle$ até ser observado

$$\frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle\,. \tag{2.2}$$

No caso da equação 2.2, o estado em superposição quando medido, resulta em $|0\rangle$ e $|1\rangle$ com cinquenta por cento de chance para cada, ou seja, $(|1/\sqrt{2}|^2)$, este estado é comumente denotado por $|+\rangle$. Enquanto as equações que representam os estados $|0\rangle$ e $|1\rangle$ respectivamente são: $|0\rangle = 1\,|0\rangle + 0\,|1\rangle$, cem porcento de chance de ser $|0\rangle$ e zero por cento de ser $|1\rangle$, e $|1\rangle = 0\,|0\rangle + 1\,|1\rangle$.

Uma boa imagem para ilustrar um qubit, é a representação geométrica dada pela esfera de Bloch demonstrada na Figura 2.1. Os números representados por θ e φ, definem o ponto unitário na esfera tridimensional e fornecem meios usuais para visualizar o estado de um único qubit.

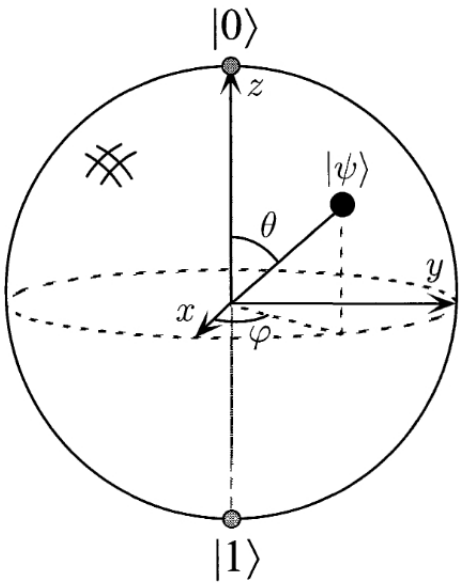

Figura 2.1 – Esfera de Bloch representando um qubit (NIELSEN; CHUANG, 2000).

Contudo, para solucionar um problema utilizando algoritmos normalmente é necessário mais de um qubit. Portanto, é fundamental entender como se comporta um sistema quântico nessa configuração. Considere 2 bits clássicos por exemplo. Com eles é possível representar os valores {00},{01},{10},{11}. Agora, suponha um sistema quântico com 2 qubits, logo, a base computacional dos estados é denotada por $|00\rangle, |01\rangle, |10\rangle, |11\rangle$. Ressalta-se que um par de qubits pode estar em estado de superposição destes quatro estados, que envolve a associação da amplitude complexa com cada base de estado computacional, tal que o vetor estado de dois qubits é descrito pela equação:

$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle\,. \tag{2.3}$$

Similar ao caso de um único qubit, o resultado da medição $x(= 00, 01, 10$ ou $11)$ ocorre com probabilidade $|\alpha_x|^2$ com os estados dos qubits após a medição sendo $|x\rangle$. A condição para que as probabilidades somem um é expressa pela condição de normalização $\sum_{x\in\{0,1\}^2} |\alpha_x|^2 = 1$, onde a notação $\{0,1\}^2$ significa que o conjunto de sequências tem comprimento dois, com cada letra podendo assumir os valores 0 ou 1.

Para um sistema de dois qubits, se possível medir apenas um subconjunto dos qubits, tem-se 0 com probabilidade $|\alpha_{00}|^2 + |\alpha_{01}|^2$, deixando o estado pós-medida da seguinte forma:

$$|\psi'\rangle = \frac{\alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}}.$$

Note, como o estado pós-medição é *renormalizado* pelo fator $\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}$, de modo que ele ainda satisfaça a condição de normalização, assim como se espera de um estado quântico. Um importante estado de dois qubits é, o estado de *Bell* ou *par EPR*,

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}.$$

O estado de Bell é responsável por muitas características interessantes na computação quântica. Esse estado tem a propriedade de após a medição do primeiro qubit, obter dois possíveis resultados: 0 com probabilidade $1/2$, e deixando o estado pós-medição $|\varphi'\rangle = |00\rangle$; e 1 com probabilidade $1/2$, deixando o estado pós medição, como $|\varphi'\rangle$ = $|11\rangle$. Como resultado, a medição do segundo qubit sempre fornece o mesmo resultado, exatamente como a medida do primeiro qubit, isto é, os resultados das medidas são *correlacionados*. Portanto, essa propriedade é fundamental na teleportação quântica e na codificação superdensa.

2.2 Operações Quânticas

Existem dois tipos de operações em qubits. Transformações unitárias e medição. O primeiro corresponde a operações reversíveis que alteram o estado de um qubit sem perder a informação e o segundo tipo, diz respeito à medição quântica. Essa operação observa o estado e obtém um valor como resultado. Note que esse processo tem como consequência que a informação contida anteriormente ao estado medido se perca (GRUSKA, 1999). Esses processos são de suma importância para o desenvolvimento de algoritmos quânticos. Nesta seção analisa-se com mais ênfase cada uma dessas transformações.

2.2.1 Transformações Unitárias

A evolução de sistemas quânticos fechados são descritas por transformações unitárias. Isto é, o estado $|\psi\rangle$ do sistema no momento t_1 está relacionado com o estado $|\psi'\rangle$ do sistema no tempo t_2 por um operador linear U que depende apenas dos tempos t_1 e t_2,

$$|\psi'\rangle = U\,|\psi\rangle\,. \tag{2.4}$$

As operações lineares são reversíveis, i.e., permitem que as informações não sejam perdidas. Alguns dos operadores lineares importantes na informação e computação quântica são as matrizes de *Pauli* X e Z. A matriz X, conhecida como porta Not Quântica pela analogia com a porta lógica clássica Not, recebe um estado $|0\rangle$ e transforma em $|1\rangle$ e $|1\rangle$ em $|0\rangle$. A matriz Z recebe um estado $|0\rangle$ e não realiza alteração, no estado $|1\rangle$ ela faz transformação para $-|1\rangle$, com o fator $-$ conhecido como *fator fase*.

Outro operador interessante é a porta *Hadamard* denotada por H. A sua ação é $H|0\rangle \equiv (|0\rangle + |1\rangle)\sqrt{2}$, $H|1\rangle \equiv (|0\rangle - |1\rangle)\sqrt{2}$, e sua matriz correspondente é:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{2.5}$$

Na computação quântica comumente refere-se como a aplicação de um operador linear em um sistema quântico particular. Por exemplo, no contexto de circuitos quânticos pode-se aplicar o operador X em um único qubit.

2.2.2 Medição Quântica

Os sistemas quânticos são fechados e evoluem de acordo com uma transformação unitária. Porém, em certos momentos é necessário observar o sistema para extrair o resultado do processamento. A Medição Quântica é justamente a observação do sistema para obter o resultado do processamento.

Em um computador clássico pode-se a qualquer momento recuperar uma informação que está contida na memória de forma simples. Por outro lado, na computação quântica quando realiza-se a medição do sistema, ocorre uma interferência em seu estado e ele entra em *colapso*, ou seja, a partir desse momento, sempre que realizada a medida do sistema o resultado irá representar sempre o mesmo valor clássico obtido, $|0\rangle$ ou $|1\rangle$ da primeira medição. Devido a isso, torna-se impossível retornar o valor anterior a medida, logo, esta é uma operação *irreversível*.

Por exemplo, quando a medida de um qubit for $|0\rangle$, a amplitude de probabilidade será $|\alpha|^2$ e $|1\rangle$ com $|\beta|^2$, logo, $|\alpha|^2 + |\beta|^2 = 1$. Note que as probabilidades devem estar normalizadas, obrigatoriamente somar 1.

Medições Quânticas são descritas pela coleção M_m de *operadores de medidas*. Esses operadores atuam no espaço estado que está sendo medido. O índice m refere-se aos resultados da medição que podem ocorrer no experimento. Se o estado do sistema quântico é $|\psi\rangle$

imediatamente após a medida, então a probabilidade do resultado m ocorrer é dada por

$$p(m) = \langle\psi| M_m^\dagger M_m |\psi\rangle , \tag{2.6}$$

e o estado do sistema após a medida é

$$\frac{M_m |\psi\rangle}{\sqrt{\langle\psi| M_m^\dagger M_m |\psi\rangle}}. \tag{2.7}$$

O operador de medida deve satisfazer a *equação de plenitude*,

$$\sum_m M_m^\dagger M_m = I.$$

A equação de plenitude expressa o fato de que a soma das probabilidades somem 1, e deve ser satisfatória para todos $|\psi\rangle$

$$1 = \sum_m p(m) = \sum_m \langle\psi \mid M_m^\dagger M_m |\psi\rangle .$$

Um simples, mas importante exemplo de medida é a *medida de um qubit na base computacional*. Esta é uma medida em um único qubit com dois resultados, definidos por dois operadores de medidas $M_0 = |0\rangle\langle 0|$ e $M_1 = |1\rangle\langle 1|$. Observe que cada operador de medida é hermitiano, portanto $M_0^2 = M_0, M_1^2 = M_1$. Suponha que o estado de medida é $|\psi\rangle = a|0\rangle + b|1\rangle$, então a probabilidade de obter a medida do resultado 0 é:

$$p(0) = \langle\psi| M_0^\dagger M_0 |\psi\rangle = \langle\psi| M_0 |\psi\rangle = \mid a \mid^2$$

Similarmente, a probabilidade de obter como resultado da medida 1 é $p(1) = |b|^2$. O estado depois da medida nos dois casos é, portanto,

$$\frac{M_0 |\psi\rangle}{|a|} = \frac{a}{|a|} |0\rangle \tag{2.8}$$

$$\frac{M_1 |\psi\rangle}{|b|} = \frac{b}{|b|} |1\rangle . \tag{2.9}$$

2.3 Circuitos Quânticos

Os circuitos quânticos são a principal forma de expressar algoritmos quânticos. A leitura do circuito é realizada da esquerda para a direita e cada linha representa um qubit do circuito quântico. É convencional assumir que o estado de entrada para o sistema seja o estado computacional base. Usualmente o estado consiste de todos os $|0\rangle$s.

A Figura 2.2 representa um circuito quântico que realiza a troca o valores de dois qubits. A sequência das portas tem os seguintes efeitos no estado base computacional $|a, b\rangle$,

$$\begin{aligned} |a, b\rangle &\longrightarrow |a, a \oplus b\rangle \\ &\longrightarrow |a \oplus (a \oplus b), a \oplus b\rangle = |b, a \oplus b\rangle \\ &\longrightarrow |b \oplus (a \oplus b), a \oplus b\rangle = |b, a\rangle \, , \end{aligned} \tag{2.10}$$

onde todas as adições $\oplus$ significam que são realizadas no módulo 2. O efeito do circuito é o intercâmbio dos estados de dois qubits.

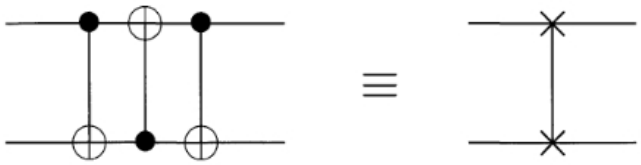

Figura 2.2 – Circuito quântico trocando dois qubits (NIELSEN; CHUANG, 2000).

Existem algumas características existentes nos circuitos clássicos que não são usualmente presentes em circuitos quânticos como *loops* e as operações FANIN e FANOUT.

Outra convenção sobre circuitos quânticos é ilustrada pela Figura 2.3. Suponha U como qualquer transformação unitária agindo sobre um número n de qubits, então U pode ser considerada uma porta quântica nesses qubits. Outro exemplo é o protótipo da U-controlada, a porta NOT-controlada que possui $U = X$ como ilustrada na Figura 2.4.

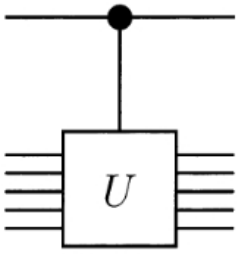

Figura 2.3 – U-Controlada (NIELSEN; CHUANG, 2000).

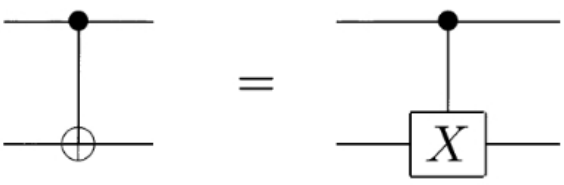

Figura 2.4 – Variação na representação da U-Controlada (NIELSEN; CHUANG, 2000).

A operação de medição é representada pelo símbolo “contador” visualizado na Figura 2.5. Esta operação converte um único estado do qubit $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ em um bit clássico probabilístico M, distinguido do qubit por uma dupla linha.

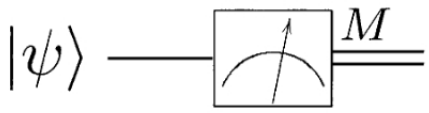

Figura 2.5 – Circuito Quântico da Medida Quântica (NIELSEN; CHUANG, 2000).

2.4 Paralelismo Quântico

Paralelismo quântico é uma característica fundamental na computação quântica. De maneira simplificada, é um método que permite um computador quântico realizar duas computações ao mesmo tempo.

Por exemplo, é possível avaliar uma função $f(x)$ para diferentes valores de x simultaneamente. Seja $f(x) : \{0,1\} \rightarrow \{0,1\}$ uma função com um bit para domínio e um para o intervalo. Uma forma de computar essa função em um computador quântico é considerar dois qubits quânticos que iniciam com os estado $|0,1\rangle$. Com uma sequência apropriada de portas lógicas é possível transformar esse estado em $|x, y \oplus f(x)\rangle$, onde $\oplus$ indica a adição de módulo 2, o primeiro registrador é o “dado” e o segundo o “alvo”. Dá-se o nome para transformação unitária que é definida pelo mapeamento $|x, y\rangle \rightarrow|\ x, y \oplus f(x)\rangle$ como U_f. Se o $y = 0$, então o estado final do segundo qubit é apenas o valor de $f(x)$.

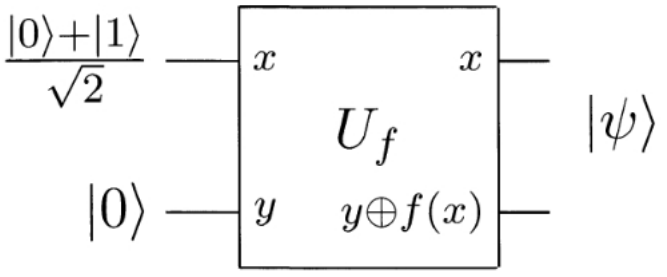

Figura 2.6 – Circuito quântico, avaliando $f(0)$ e $f(1)$ simultaneamente (NIELSEN; CHUANG, 2000)

.

Considere o circuito visualizado na Figura 2.6 que aplica U_f para a entrada NOT na base

computacional, com o dado registrador preparado na superposição $|1\rangle/\sqrt{2}$, que pode ser criado com a porta Hadamard agindo no $|0\rangle$, então aplicando U_f o resultado é o seguinte estado:

$$\frac{|\,0, f(0)\rangle + |\,1, f(1)\rangle}{\sqrt{2}}. \tag{2.11}$$

Esse é um estado notável, pois contém informação referente aos dois estados $f(0)$ e $f(1)$. Ou seja, é como se a avaliação de $f(x)$ tenha sido realizada simultaneamente, determinando o paralelismo quântico.

Pode-se generalizar esse procedimento para funções em um numero arbitrário de bits, utilizando a operação *Transformada de Hadamard*. Esta operação é composta por n portas Hadamard agindo em paralelo em n qubits. Para dois qubits, ou seja, $n = 2$, a preparação para o estado $|0\rangle$ tem como saída:

$$\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) = \frac{|00\rangle + |01\rangle + |10\rangle + |11\rangle}{2}. \tag{2.12}$$

Denota-se $H^{\otimes 2}$ como a ação paralela em duas portas Hadamard e $\otimes$ representa o produto tensorial. De forma geral, o resultado da execução da transformada de Hadamard no estado em n qubits inicialmente no estado $|0\rangle$ é

$$\frac{1}{\sqrt{2^n}} \sum_x |x\rangle \tag{2.13}$$

onde a soma é sobre todas as possibilidades dos valores de x denotada por $H^{\otimes}$. Isto é, a transformada Hadamard produz uma igual superposição de todos os estados bases computacionais. Além disso, é extremamente eficiente produzindo a superposição de 2^n estados usando apenas n portas.

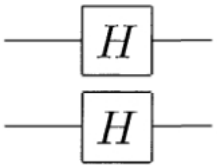

Figura 2.7 – Transformada Hadamard $H^{\oplus 2}$ em dois qubits (NIELSEN; CHUANG, 2000).

A avaliação quântica paralela de uma função com n bits de entrada e um bit de saída, $f(x)$ pode ser realizada da seguinte forma. Prepara-se $n + 1$ estados do qubit $|0\rangle^{\oplus n} |0\rangle$ e então aplica-se a transformada de Hadamard para os primeiros n qubits, em seguida implementado o circuito quântico U_f, com isso será produzido o estado

$$\frac{1}{\sqrt{2^n}} \sum_x |x\rangle \, |f(x)\rangle \,. \tag{2.14}$$

Da mesma forma, o paralelismo quântico avalia todas as possibilidades de valores da função f simultaneamente, apesar de aparentemente parecer apenas uma avaliação de $f(x)$.

2.5 Algoritmos Quânticos

Para resolver um problema particular, os computadores, sejam clássicos ou quânticos, devem seguir um conjunto preciso de instruções mecanicamente, com o objetivo de produzir a solução para qualquer instância do problema. Esse conjunto de instruções é conhecido como *algoritmo* (EKERT; HAYDEN; INAMORI, 2000).

Algoritmos quânticos podem ser executados em qualquer modelo realístico da computação quântica. A notação comumente utilizada para a descrição de algoritmos quânticos é através de circuitos quânticos. Existem outros modelos de descrição de algoritmos para computação quântica, sendo que, circuitos quânticos podem simular qualquer outro tipo de notação, além de ser a maneira mais explorada pelo meio científico.

Algoritmos, portanto, são notações necessárias para descrever o comportamento do processamento de informações tanto clássicas como quânticas. As técnicas de programação para computação clássica podem ser consideradas maduras por tratarem-se de tecnologias consolidadas. Por outro lado, o paradigma voltado para a computação quântica está começando a ser explorado com mais ênfase nos últimos anos, sendo que, a tendência é de que cada vez mais sejam despendidos esforços no seu desenvolvimento.

Notavelmente a computação quântica possui atrativos convincentes, como o ganho no poder de processamento computacional, que se comparado com um computador clássico, se dá na ordem crescente exponencial. Somente com esse aumento, a noção do que são problemas tratáveis e intratáveis acabariam tendo nova proporção, ou seja, alguns problemas antes impossíveis de resolver em um tempo satisfatório, podem vir a serem resolvidos em tempo polinomial.

Portanto, o estudo de algoritmos quânticos é importante por diversas razões, sendo que sua aplicação poderá abranger uma vasta quantidade de áreas de conhecimento, por exemplo: segurança computacional, previsão do tempo, estudo genético, espacial, medicina, enfim, diversas áreas conhecidas e exploradas em nossa sociedade.

São descritos os seguintes algoritmos nas próximas seções: os algoritmos da teleportação e codificação superdensa por tratarem-se de protocolos de informação quântica que são contemplados pela Teoria Quântica Modal, e o algoritmo de Deutsch que determina o mapea-

mento de valores para uma dada função $f(x)$.

2.5.1 Algoritmo da Teleportação Quântica

Pode-se definir a teleportação quântica como o processo que transfere o estado de um qubit para outro qubit. Por exemplo, deseja-se realizar a transmissão de um estado quântico, para isso é necessário conhecer as posições e velocidades de todos os elétrons, porém pelo princípio da incerteza de Heisenberg uma partícula não possui posição e velocidade simultaneamente bem definidas, ou seja, a teleportação seria impossível. No entanto, Charles Bennett mostrou como utilizar as propriedades dos objetos quânticos remontando o paradoxo de Einstein, Podolsky e Rosen (EPR) (BENNETT et al., 1993).

O autor ainda ressalta que na prática, considerando um par de fótons, conhecendo o plano de polarização em um dos fótons, o segundo fica automaticamente determinado como sendo perpendicular ao primeiro, sem necessidade de observá-lo. Entretanto, existe uma propriedade referente ao estado quântico, ele não pode ser conhecido sem que haja um processo de observação, ou seja, até lá o estado permanece em uma sobreposição dos dois estados possíveis.

Pode-se dizer que houve uma certa comunicação instantânea entre os fótons com velocidade superior à da luz. Mas no entanto esta é uma situação impossível, pois, segundo Einsten contradiz a lei da Relatividade. Contudo, o efeito EPR é real, Bell realizou previsões baseadas neste efeito, conhecida como as famosas desigualdades de Bell e, os resultados obtidos mostraram-se de acordo com estas previsões, ou seja, não há comunicação entre os fótons, i.e., até eles serem observados são um único sistema quântico (BELL, 1964).

O exemplo geralmente utilizado para a exemplificação do algoritmo de teleporte quântico é o seguinte: suponha que os observadores chamados de Alice e Bob compartilham um par emaranhado. Alice tem como objetivo enviar um qubit, $|\psi\rangle$, para Bob, mas ela não sabe o estado desse qubit, e não pode realizar um processo de medida no sistema, pois ele entrará em colapso e as informações se perderão.

Os passos para a solução do problema são: Alice interage o qubit com sua metade do par EPR, então realiza o processo de medição nos dois qubits que ela possui, obtendo uma dos quatro resultados possíveis 00, 01, 10 e 11. Então ela envia esta informação para Bob. Dependendo da mensagem clássica de Alice, Bob realiza uma das quatro operações em sua metade do par EPR e consegue recuperar o estado de $|\psi\rangle$.

O circuito quântico que representa algoritmo de teleportação pode ser visualizado na

Figura 2.8. Onde o estado a ser teleportado é $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, α e β são escalares de probabilidade desconhecidos. O estado de entrada do sistema é $|\psi_0\rangle$:

$$|\psi_0\rangle = |\psi\rangle\,|\beta_{00}\rangle \tag{2.15}$$

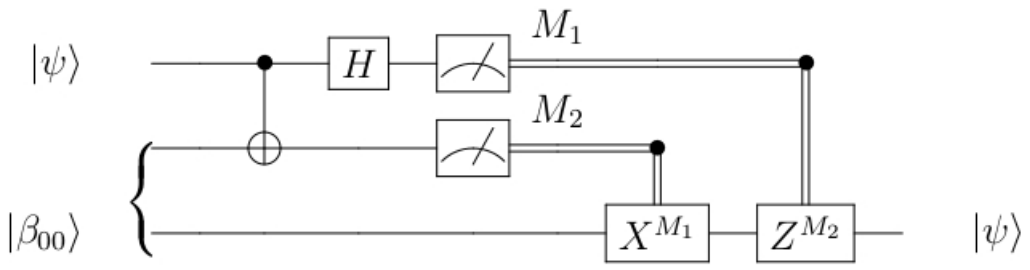

Figura 2.8 – Circuito quântico do algoritmo da teleportação (NIELSEN; CHUANG, 2000).

As duas primeiras linhas representam o sistema quântico de Alice e a terceira de Bob. As linhas duplas denotam a transmissão de bits clássicos e as simples representam as transmissões dos qubits. A saída do sistema é:

$$= \frac{1}{\sqrt{2}}\,[\alpha|0\rangle\,(|00\rangle + |11\rangle) + \beta|1\rangle\,(|00\rangle + |11\rangle)]\,. \tag{2.16}$$

Por convenção, o primeiro e o segundo qubits pertencem a Alice e o terceiro a Bob. Então Alice envia seu qubit através de uma porta `CNOT`, obtendo:

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}\,[\alpha|0\rangle\,(|00\rangle + |11\rangle) + \beta|1\rangle\,(|10\rangle + |01\rangle)]\,. \tag{2.17}$$

Se Alice envia o primeiro qubit através de uma porta Hadamard, obtém-se:

$$|\psi_2\rangle = \frac{1}{2}\,[\alpha(|0\rangle + |1\rangle)(|00\rangle + |11\rangle) + (\beta|0\rangle - |1\rangle)(|10\rangle + |01\rangle)]\,. \tag{2.18}$$

Este estado pode ser escrito reagrupando os termos, da seguinte forma:

$$\begin{aligned}|\psi_2\rangle = \frac{1}{2}\,[&|00\rangle\,\alpha(|0\rangle + \beta|1\rangle) + (|01\rangle + \alpha|1\rangle + \beta|0\rangle) + \\ &|10\rangle\,\alpha(|0\rangle - \beta|1\rangle) + (|11\rangle + \alpha|1\rangle - \beta|0\rangle)].\end{aligned} \tag{2.19}$$

Essa expressão naturalmente se decompõe em quatro estados. O primeiro termo tem o qubit de Alice no estado $|00\rangle$ e o qubit de Bob no estado $\alpha|0\rangle + \beta|1\rangle$ que é o estado original de $|\psi\rangle$. Se Alice realizar a medição e obtiver o resultado 00, então o sistema de Bob vai ser $|\psi\rangle$.

De forma similar, a partir da expressão anterior, o estado pós-medição de Bob dado o resultado da medição de Alice são os seguintes estados:

$$00 \mapsto |\psi_3(00)\rangle \equiv [\alpha\,|0\rangle + \beta\,|1\rangle] \tag{2.20}$$

$$01 \mapsto |\psi_3(01)\rangle \equiv [\alpha\,|1\rangle + \beta\,|0\rangle] \tag{2.21}$$

$$10 \mapsto |\psi_3(10)\rangle \equiv [\alpha\,|0\rangle - \beta\,|1\rangle] \tag{2.22}$$

$$11 \mapsto |\psi_3(11)\rangle \equiv [\alpha\,|p\rangle - \beta\,|0\rangle]. \tag{2.23}$$

Portanto, dependendo do resultado da medição de Alice, o estado de Bob irá para um destes quatro estados possíveis. Por exemplo, no caso que a medida produzir 00, Bob não precisa fazer nada. Se a medida for 01, então Bob pode reproduzir o seu estado, aplicando a porta X. Se a medição é 10, Bob pode reproduzir seu estado, aplicando o porta Z. Por fim, se a medição é de 11, Bob pode corrigir seu estado, aplicando primeiro a porta X e depois Z. Resumindo, Bob precisa aplicar a transformação $Z^{M_1}X^{M_2}$ em seu estado (note como o tempo passa da esquerda para a direita nos diagramas de circuitos, mas em produtos de matriz os termos à direita ocorrem primeiro) para o seu qubit, sendo assim, e ele vai recuperar o estado $|\psi\rangle$.

2.5.2 Algoritmo da Codificação Superdensa

A codificação superdensa é um protocolo que mostra uma forma de como pode-se transmitir dois bits clássicos utilizando apenas um qubit. Este resultado foi obtido por (BENNETT; WIESNER, 1992), um dos primeiros exemplos de que o emaranhamento pode facilitar a comunicação.

O seu funcionamento é da seguinte forma. Suponha que Alice e Bob compartilham o estado $|\beta_{00}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, sendo o primeiro qubit pertencente a Alice e o segundo a Bob. A primeira parte do protocolo consiste em, Alice realizar operações unitárias em seu qubit, de acordo com a mensagem que deseja enviar.

- Se Alice deseja enviar $x_1 = 0$ e $x_2 = 0$, então ela não realiza nenhuma operação e o resultado final obtido é igual a $|\beta_{00}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$
- Se Alice deseja enviar $x_1 = 0$ e $x_2 = 1$, ela aplica Z no seu qubit e o estado final resultante é $|\beta_{01}\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$

- Se Alice deseja enviar $x_1 = 1$ e $x_2 = 0$, ela aplica X no seu qubit e o estado final resultante é $|\beta_{10}\rangle = \frac{1}{\sqrt{2}}(|10\rangle + |01\rangle)$
- Se Alice deseja enviar $x_1 = 1$ e $x_2 = 1$, ela aplica XZ no seu qubit e o estado final resultante é $|\beta_{11}\rangle = \frac{1}{\sqrt{2}}(|10\rangle - |01\rangle)$.

Observe que os estados $|\psi_{x_1x_2}\rangle$ são ortogonais entre si, ou seja, eles sempre podem ser discriminados. Alice envia o qubit para Bob, então Bob possuindo o estado $|\psi_{x_1x_2}\rangle$ pode realizar uma medição com os projetores de Bell $\{|\beta_{00}\rangle\langle\beta_{00}|, |\beta_{01}\rangle\langle\beta_{01}|, |\beta_{10}\rangle\langle\beta_{10}|, |\beta_{11}\rangle\langle\beta_{11}|\}$, e utilizar o resultado para obter x_1 e x_2.

2.5.3 Algoritmo de Deutsch

Uma modificação do circuito demonstrado na Figura 2.6 demonstra como circuitos quânticos podem realizar uma implementação clássica do algoritmo de *Deutsch*. Esse algoritmo mistura uma combinação de paralelismo quântico e uma propriedade da Mecânica Quântica, chamada de interferência. Utiliza-se a transformada Hadamard pra preparar o primeiro qubit com a superposição $|0\rangle + |1\rangle / \sqrt{2}$, e prepara-se o segundo qubit com $|0\rangle - |1\rangle / \sqrt{2}$, utilizando a Hadamard aplicada ao estado $|1\rangle$. Aplicando a entrada $|\psi\rangle|01\rangle$, comportamento do algoritmo é descrito pela Figura 2.9.

Estado de entrada:

$$|\psi_0\rangle = |01\rangle \tag{2.24}$$

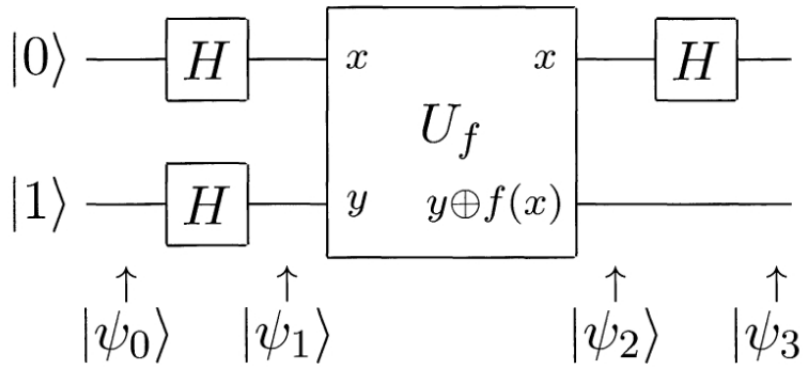

Figura 2.9 – Circuito quântico implementando o algoritmo de Deutsch (NIELSEN; CHUANG, 2000).

A saída do estado é determinada por

$$|\psi_1\rangle = \left[\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{2.25}$$

Se aplicar U_f ao estado $(|0\rangle - |1\rangle)/\sqrt{2}$ então obtém-se $(-1)^{f(x)}|x\rangle(|0\rangle - |1\rangle)/\sqrt{2}$, aplicando U_f em $|1\rangle$, portanto nos leva a duas possibilidades:

$$|\psi_2\rangle = \begin{array}{ll} \pm\left[\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] & \text{if } f(0) = f(1) \\ \pm\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] & \text{if } f(0) \neq f(1) \end{array} \tag{2.26}$$

A porta final Hadamard no nosso primeiro qubit é dada por:

$$|\psi_3\rangle = \begin{array}{ll} \pm\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] & \text{if } f(0) = f(1) \\ \pm\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] & \text{if } f(0) \neq f(1). \end{array} \tag{2.27}$$

Portanto, realizando $f(0) \oplus f(1)$ é 0, se $f(0) = f(1)$ e 1, então pode-se reescrever o resultado como:

$$|\psi_3\rangle = \pm|f(0) \oplus f(1)\rangle \pm \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right], \tag{2.28}$$

medindo o primeiro estado determina-se o $f(0) \oplus f(1)$. Esse fato é muito importante, porque o circuito quântico tem a habilidade de determinar uma *propriedade global* de $f(x)$, nomeado $f(0) \oplus f(1)$ utilizando apenas *uma* avaliação de $f(x)$. Por esta razão, esse processo é mais rápido que um dispositivo clássico, que deve obrigatoriamente realizar duas avaliações.

Este exemplo mostra a diferença entre o paralelismo quântico e clássico. De forma equivocada pode-se pensar que o estado $|0\rangle|f(0)\rangle + |1\rangle|f(1)\rangle$ corresponde mais próximo a um computador clássico probabilístico que avalia $f(0)$ ou $f(1)$ com 50% de probabilidade.

A diferença é que em um modelo clássico, essas duas alternativas sempre excluem uma a outra, enquanto que em um computador quântico para ambas possibilidades as alternativas *interferem* uma com a outra para algumas propriedades globais da função f e a porta Hadamard combina as diferentes alternativas que foram realizadas no algoritmo.

2.6 Resumo do Capítulo

Neste capítulo apresenta-se uma série de conceitos presentes na computação quântica que são oriundos da Mecânica Quântica. O entendimento do comportamento de bits quânticos e a manipulação do seu estado são fatores de suma importância no desenvolvimento de algoritmos. Além disso, nota-se que a comparação entre a computação clássica e a quântica é um fato inevitável, pois a computação clássica além de ser a tecnologia atual, é a base para o modelo a desenvolvido. Por fim, se houver a necessidade ou interesse em aprofundar os conhecimentos

conceituais da computação quântica, pode-se citar o livro *Quantum Computation and Quantum Information* (NIELSEN; CHUANG, 2000), o artigo *Teleporting an Unknow Quantum State via Dual Classical and Einstein-Podolsky-Rosen Chanels* (BENNETT et al., 1993), e o livro *Feynman Lectures on Computation* (FEYNMAN, 2000).

3 TEORIA QUÂNTICA MODAL

Como visto no capítulo anterior, na física quântica convencional, um vetor no espaço de Hilbert descreve o estado de um sistema quântico. Associados ao estado, os escalares α e β representam a probabilidade de estar em um dos estados básicos $|0\rangle$ ou $|1\rangle$. Os escalares de probabilidade são descrito por um número complexo $\mathbb{C}$. Contudo, números complexos, por serem infinitos, são incomputáveis (NIELSEN; CHUANG, 2000).

A Teoria Quântica Modal (TQM) é um modelo de CQ alternativa, apresentado por Schumacher e Wostemoreland (2011), que possui como principal característica a *substituição de escalares complexos por campos finitos*, resultando em um modelo intermediário de Computação Quântica.

A primeira mudança fundamental ocorre na estrutura matemática envolvida na TQM. Em campos finitos não existem as noções de produto interno e ortogonalidade, ambas que são fundamentais no formalismo do espaço de Hilbert. No entanto, apesar de ser uma teoria mais simples se comparada com o formalismo matemático da Computação Quântica atual, esta teoria conserva muito da estrutura e das características da Mecânica Quântica, sendo que pode-se citar como exemplo os seguintes aspectos:

- as noções de superposição, interferência, emaranhamento e estados mistos de sistemas quânticos;
- a evolução temporal do sistema quântico utiliza operadores lineares reversíveis;
- a complementaridade dos observáveis incomputáveis;
- a exclusão da variável local escondida e a impossibilidade de clonagem de estados quânticos;
- a presença de contrapartes naturais de informação quântica e os protocolos da codificação superdensa e teletransporte quântico.

A representação da Teoria Quântica Modal é realizada a partir de um campo finito binário $\mathbb{F}_2$ de booleanos. Ao contrário da interpretação da CQ convencional que envolve probabilidades qualitativas, a CQ Modal abrange apenas a distinção entre *possível* e *impossível* comumente representadas respectivamente por $True$ e $False$.

Para um campo finito $\mathbb{F}$, o estado de um sistema é um vetor não nulo $|\psi)$ em um espaço vetorial de dimensão finita $\mathbb{V}$ que é isomórfico a $\mathbb{F}^d$ para alguma dimensão d. Ou seja, na estrutura resultante, todos os vetores não nulos representam estados quânticos válidos, sendo que, a evolução de um sistema quântico fechado é descrita por meio de mapas lineares reversíveis arbitrários.

Por exemplo, seja $\mathbb{C}$ um conjunto de resultados de um experimento, para que ele seja possível (sendo a distinção entre possível e impossível tal como na lógica modal) o resultado da medição de um sistema, gera um subconjunto $\mathbb{P} \subseteq \mathbb{C}$, logo, se o valor estiver no subconjunto $\mathbb{P}$ é possível, caso contrário impossível.

O sistema mais simples é determinado por um campo finito $\mathbb{Z}_2$, denominado de mobit. Um mobit pode ter os estados básicos $|0)\,, |1)$ e o estado em superposição $|\sigma) = |0) + |1)$. Os estados são dependentes lineares, portanto, pode-se construir os estados de um mobit a partir da superposição dos outros dois estados, ou seja:

$$\begin{aligned} |0) &= |1) + |\sigma) \\ |1) &= |0) + |\sigma) \\ |\sigma) &= |0) + |1) \end{aligned} \tag{3.1}$$

Ressalta-se que estruturas discretas de computações também já foram utilizadas por outros modelos de computação quântica, pode-se citar os trabalhos *Discrete Quantum Theories* (HANSON et al., 2013) que utiliza campos complexos finitos; e em *Quantum Computing over Finite Fields: Reversible Relational Programming with Exclusive Disjunctions* James, Ortiz e Sabry (2011), que utiliza campos finitos para a construção de uma metalinguagem utilizando a linguagem Prolog.

A TQM apresenta uma teoria extremamente simplificada definida através de campos finitos arbitrários sem a noção de unitariedade. Entretanto, essa simplicidade cria uma teoria com superpoderes anormais permitindo, por exemplo, a busca em base de dados não estruturadas mais rápida que assimptoticamente possível na computação quântica convencional (HANSON et al., 2011).

O desenvolvimento deste trabalho esta fortemente baseado na TQM. Para facilidade de discernimento utiliza-se a nomenclatura **mobit** (*Modal Qubit*) para fazer relação com o bit quântico, e a notação $|\psi)$ substituindo a notação $|\psi\rangle$ *ket* da computação quântica convencional.

O restante do capítulo está organizado da seguinte forma: A seção 3.1 apresenta a estrutura matemática de campos finitos, que será a base para construção do Modelo Quântico Modal. A seção 3.2 apresenta os fundamentos dos espaços vetoriais da TQM. Na seção 3.3 demonstra-

se como ocorre a evolução temporal dos sistemas quânticos modais. A seção 3.4 faz referência a estados emaranhados na Teoria Quântica Modal. A seção 3.5 descreve o processo de medição quântica considerando a TQM. Nas seções 3.6 e 3.7 são apresentados respectivamente a resolução dos algoritmos da Codificação Superdensa e UNIQUE-SAT. Para finalizar, a seção 3.8 apresenta um breve resumo do capítulo com os pontos principais.

3.1 Campos Finitos

Segundo Mcelliece (1987), campos finitos são abstrações de sistemas numéricos matemáticos, tal como o conjunto de números complexos $\mathbb{C}$ e reais $\mathbb{Q}$ por exemplo. Campos finitos consistem em um conjunto $\mathbb{F}$, juntamente com duas operações, adição (denotada por +) e multiplicação (denotada por $\cdot$). A estrutura de um campo finito tem as seguintes características:

- Para cada a no campo, $a \cdot 0 = 0 \cdot a = 0$.
- Para qualquer elemento não nulo a e b no campo, $a \cdot b \neq 0$.
- $a \cdot b = 0$ e $a \cdot b \neq 0$ implica que $b = 0$.
- $(\mathbb{F}, +)$ é um grupo abeliano com identidade denotado por zero.
- $(\mathbb{F} \ \{0\}, \cdot)$ é um grupo abeliano com multiplicativo denotado pela identidade 1.
- A lei distributiva mantém: $(a + b) \cdot c = a \cdot c + b \cdot c$ 1 para todo $a, b, c \in \mathbb{F}$.

Um campo finito é equipado com duas operações, multiplicação e adição. Ressalta-se que a subtração é definida em termos da operação de adição. Por exemplo: para $a, b \in \mathbb{F}, a-b = a + (-b)$ onde $-b$ é o único elemento em $\mathbb{F}$, tal que $b + (-b) = 0$ ($-b$ é chamado o negativo de b). Similarmente, a divisão entre elementos de um campo finito é definida em termos da multiplicação: para $a, b \in \mathbb{F}$ com $b \neq 0$, $a \div b = a \cdot b^{-1}$, onde b^{-1} é o único elemento em $\mathbb{F}$ tal que $b \cdot b^{-1} = 1$ e b^{-1} é chamado de *inverso* de b.

Portanto, campos finitos são basicamente um conjunto de elementos onde aplicam-se as operações de soma, subtração, multiplicação e divisão sem sair da gama desse conjunto, logo, são necessariamente *cíclicos*. Além disso, existem outras características relevantes em campos finitos: se o conjunto $\mathbb{F}$ é finito, então o campo é dito finito; o número de elementos é denotado como a ordem do campo; conhecido também como *Galois Fields*; um campo finito de ordem p pode ser denotado como $GF(p)$.

O campo finito $\mathbb{F}_2$ de booleanos consiste em apenas dois valores base *True* e *False*. A operação de adição é semelhante à porta *or-exclusive* e à multiplicação semelhante a uma conjunção. As operações de soma e multiplicação em um campo $\mathbb{F}_2$ podem ser expressas da seguinte forma:

$$\begin{array}{ll} 0+0=0 & 0\cdot 0=0 \\ 0+1=1 & 0\cdot 1=0 \\ 1+0=1 & 1\cdot 0=0 \\ 1+1=0 & 1\cdot 1=1 \end{array} \tag{3.2}$$

Estas características são intuitivamente consistentes com a interpretação de escalares como probabilidades em eventos quânticos, exceto o fato de que $1+1$ obtém-se 0, ou seja, ter um evento duas vezes possível que torna-se impossível.

3.2 Espaços Vetoriais

O caso mais simples é a construção de um mobit que possui os estados base $|0)$ e $|1)$, o conjunto limite de estados possíveis é definido pelos coeficientes α e β que pertencem a um conjunto finito. Portanto, considerando um campo $\mathbb{F}_2$, existem exatamente três estados válidos para um mobit: $1\,|0)+0\,|1)$ equivalente ao estado $|0)$, o estado $0\,|0)+1\,|1)$ equivalente a $|1)$ e por fim o estado em superposição $1\,|0)+1\,|1)$ denotado por $|\sigma)$.

Os estados são base dependentes, por isso, as bases modais definidas para um mobit podem ser representadas por X, Y e Z

$$\begin{array}{lll} |+_x)=|1) & |+_y)=|\sigma) & |+_z)=|0) \\ |-_x)=|\sigma) & |-_y)=|0) & |-_z)=|1) \end{array}. \tag{3.3}$$

Por outro lado, para um par de mobits temos a seguinte base: $\{|00)\,|01)\,,|10)\,,|11)\}$, excluindo o vetor zero, é possível formar quinze estados válidos, sendo entre eles seis produto

estado e nove estados emaranhados

$$
\begin{gathered}
|0) \oplus |0) = |0,0) \\
|0) \oplus |1) = |0,1) \\
|0) \oplus |0) + |1) = |0,0) + |0,1) \\
|1) \oplus |0) = |1,0) \\
|1) \oplus |1) = |1,1) \\
|1) \oplus |0) + |1) = |1,0) + |1,1) \\
|0) + |1) \oplus |0) = |0,0) + |1,0) \\
|0) + |1) \oplus |1) = |0,1) + |1,1) \\
|0) + |1) \oplus |0) + |1) = |0,0) + |0,1) + |1,0) + |1,1) \\
|0,0) + |1,1) \\
|0,1) + |1,0) \\
|0,0) + |0,1) + |1,0) \\
|0,0) + |1,0) + |1,1) \\
|0,0) + |0,1) + |1,1) \\
|0,1) + |1,0) + |1,1)\,.
\end{gathered}
\tag{3.4}
$$

Os estados produto, são os estados que podem ser produzidos a partir do produto tensorial de outros estados, enquanto que em estados emaranhados não pode-se atribuir um estado independente puro a cada uma das partes, é comum dizer que ambas as partes possuem estado físico próprio.

3.3 Evolução Temporal dos Sistemas Quânticos Modais

Na computação quântica modal, a evolução temporal de um sistema pode ser considerada como uma sequência de intervalos discretos, representada por uma transformação linear T do vetor de estado. Assim se $|a) \rightarrow |a') = T\,|a)$ e $|b) \rightarrow |b') = T\,|b)$ então $|a) + |b) \rightarrow T(|a) + |b)) = T\,|a) + |b) = |a') + |b')$.

O vetor zero não é um estado físico, pois é necessário que $T\,|a) \neq 0$ para qualquer estado $|a)$. Isto significa que o núcleo de T é trivial, de modo que T é reversível.

Sem nenhuma restrição adicional, a propriedade da evolução de sistemas discretos deve considerar que qualquer transformação linear inversível de vetores de estados corresponde a uma possível evolução do estado do sistema.

3.4 Estados Emaranhados

Na mecânica quântica, o produto tensorial descreve um sistema composto no espaço de Hilbert. Essa mesma regra pode ser aplicada a espaços vetoriais na computação quântica modal. Em geral, sistemas compostos podem ser produto estados e não-produto (emaranhado).

No espaço estado modal, pode-se calcular o número de produtos e estados emaranhados para um dado par de sistemas.

Considere um par de mobits tal que $F = \mathbb{Z}_2$. Como visto na Seção anterior, existem 15 estados vetoriais permitidos expressos pela Equação 3.4 (excluindo o vetor nulo), todos representando estados distintos do sistema. O estado emaranhado de dois mobits $|S) = |0,1) + |1,0)$ tem a seguinte propriedade: qualquer efeito do produto $(a,a|$ é impossível para $|S)$ devido a Equação 3.5.

$$(a,a|\, S) = (a\,|0)\,(a\,|1) + (a\,|0)\,(a\,|0) = 0 \tag{3.5}$$

Com isso pode-se chegar às seguintes conclusões:

- Se a mesma medição é criada em cada mobit, então apenas é possível unir resultados que tem valores opostos para cada mobit, por exemplo, $(+_z, -_z)$ é possível; o resultado $(+_z, +_z)$ é impossível pois $(+_z, +_z| = 0$.
- Se medições diferentes são criadas em mobits, então existe uma união que é impossível. Por exemplo, $(+_z, -_x)$ é impossível pois $(+_z, -_x) = (0,0|$.

A correlação de incompatibilidade entre sistemas quânticos emaranhados com qualquer teoria da variável escondida foi proposta por Bell (BELL, 1964). Infelizmente, não existe um comportamento equivalente na TMQ devido à ausência de probabilidades e expectativa de valores. Contudo Hardy (HARDY, 1993) concebeu uma aproximação baseada apenas na possibilidade e impossibilidade.

Ele construiu um estado não-totalmente emaranhado $|\psi)$ de um par de mobits, junto com observáveis binários **A** e **B** em cada mobit, denotada por $(x, y \mid X, Y)$ o resultado (x, y) de uma medição conjunta (X, Y), então o estado de Hardy tem as seguintes propriedades:

- $(0,0 \mid A, B)$ e $(0,0 \mid B, A)$ é impossível, pois eles possuem probabilidade $p = 0$
- $(0,0 \mid B, B)$ é possível pois $(p > 0)$
- $(1,1 \mid A, A)$ é impossível pois $(p = 0)$

3.5 Medição Quântica Modal

No padrão da MQ convencional (baseada no espaço de Hilbert), o postulado da medição requer a noção de projeção ortogonal. Contudo, investigou-se que na teoria quântica modal não

existe o conceito de produto interno e, portanto, também a noção de ortogonalidade. Apesar disso, é possível definir um processo de medição quântica intermediário, agora baseada nas características de campos finitos.

No processo da medição quântica modal, pode-se citar como pontos principais duas características, (i) qualquer conjunto de vetores linearmente independentes pode formar uma base, (ii) é possível definir dois vetores para uma dada base.

Partindo desse conceito (considere um campo finito $\mathbb{F}$) o estado do sistema é um vetor não-nulo $|\psi)$ em um espaço vetorial dimensional finito $\mathbb{V}$ que é isomórfico a $\mathbb{F}^d$ de dimensão d. A medição do sistema corresponde ao conjunto base $A = \{|a)\}$ para $\mathbb{V}$, onde cada elemento $|a)$ é associando com um resultado a do procedimento de medição. Portanto, todo vetor estado pode ser escrito como:

$$|\psi) = \sum_a \psi_a \, |a) \tag{3.6}$$

onde ψ_a são os escalares em $\mathbb{F}$. O resultado da medição é possível se e apenas se $\psi_a \neq 0$. Para uma base A e o estado ψ, o conjunto de possíveis resultados da medição é descrito por

$$P(A \, |\psi) = \{a : \psi_a \neq 0\}. \tag{3.7}$$

Sabendo que um mobit possui três estados possíveis $|0) \, , |1)$ e $|\sigma)$ e que os estados são base dependente, logo, pode-se definir um estado a partir da superposição dos outros dois, por exemplo o mobit $|1)$ pode ser obtido a partir superposição dos estados $|0) + |\sigma)$ e assim por diante.

Para saber se um elemento da base pode ser um resultado possível de medição, é necessário considerar a base completa. Considerando a base Z, para o estado $|\sigma)$, utilizando a Equação 3.7 pode-se facilmente visualizar que os estados de medições possíveis são:

$$\begin{gathered} P(Z \mid \sigma) = \{(|0) \, , |1)\} \\ P(Z \mid 0) = \{|0)\} \\ P(Z \mid 1) = \{|1)\}. \end{gathered} \tag{3.8}$$

Neste exemplo, a Equação 3.7 é eficaz para o processo de medida. Entretanto, ao aplicar a mesma equação à base Y, o elemento da base $|0)$ não é um resultado possível. Portanto, é necessário associar a medição com uma base do espaço dual V^*. Assim, dada a base A para V^* tal que $(a| \, \psi) = \psi_a$, na falta de produto interno a correspondência entre $(a|$ e $|a)$ é base dependente. Os funcionais em V^* são chamados *efeitos*, e é dito que um efeito é possível se e apenas se fornece $(a| \, \psi) \neq 0$. Portanto, dada uma base A para V^*,

$$P(A \, |\psi) = \{a : (a \, |\psi) \neq 0\}. \tag{3.9}$$

Considerando as três bases X, Y e Z (demonstradas na Equação 3.3) possíveis para um mobit é necessário considerar a base dual.

As base duais são obtidas a partir da operação: $(0|\,0) = (1|\,1) = 1$ e $(1|\,0) = (0|\,1) = 0$. Por exemplo, para gerar a base dual para a base Z = $\{|0),|1)\}$, deve-se levar em consideração o estado $|\sigma)$, uma vez que, esse é o estado complementar do mobit. Para visualizar a definição da base dual temos a seguinte equação:

$$\begin{gathered} P(Z\,|\sigma) = \{a : (a\,|\psi) \neq 0\}. \\ P\,(0|\,\sigma) = (0 \mid 0) + |1) = (0 \mid 0) + (0 \mid 1) = 1 + 0 = 1 \\ P\,(1|\,\sigma) = (1 \mid 0) + |1) = (1 \mid 0) + (1 \mid 1) = 0 + 1 = 1 \\ P\,(\sigma|\,\sigma) = (0| + (1| \mid |0) + |1) = (0 \mid 0) + (0 \mid 1) + (1 \mid 0) + (1 \mid 1) = 1 + 0 + 0 + 1 = 0. \end{gathered} \tag{3.10}$$

Aplicando esta equação para os demais estados, temos as bases duais X, Y e Z :

$$\begin{array}{lll} (+_x\,| = (\sigma \mid & (+_y\,| = (1 \mid & (+_z\,| = (0 \mid \\ (-_x\,| = (0 \mid & (-_y\,| = (\sigma \mid & (-_z\,| = (1 \mid \end{array}. \tag{3.11}$$

Considerando as bases duais, para medir o estado $|\sigma)$ na base Y deve-se considerar a base dual, utilizando a Equação 3.9. Os resultados possíveis para os elementos da base $|\,+_y)$ e $|\,-_y)$ são

$$\begin{gathered} P(+_y\,|\sigma) = (1\,|0) + |1) = \\ (1\,|0) + (1\,|1) = \\ 0 + 1 = 1 \end{gathered} \tag{3.12}$$

$$\begin{gathered} P(-_y\,|\sigma) = (0 \mid +(1\,|0) + |1) = \\ (0\,|0) + (0\,|1) + (1\,|0) + (1\,|1) = \\ 1 + 0 + 1 + 0 = 0. \end{gathered} \tag{3.13}$$

Note que para um resultado da base ser possível, é necessário que $\psi_a \neq 0$. Por isso, pode-se visualizar nas Equações 3.12 e 3.13 que o conjunto possível de resultados da medição para esta base é $P(Y\,|\sigma) = (1|$.

3.6 Algoritmo da Codificação Superdensa

Uma característica interessante de estados emaranhados na teoria quântica convencional é a codificação superdensa. Esta propriedade permite que através do emaranhamento seja possível dobrar a capacidade de informações contida em um sistema quântico. Na TMQ existe uma versão do protocolo da codificação superdensa.

Para a entender esse algoritmo deve-se considerar os seguintes estados bases para 2 qubits:

$$\begin{array}{ll} |R) = |0,0) + |1,1) & |U) = |0,0) + |1,0) + |1,1) \\ |S) = |0,1) + |1,0) & |V) = |0,0) + |0,1) + |1,0) \end{array} . \tag{3.14}$$

Estes quatro estados emaranhados formam uma base e também podem ser identificados como um resultado de medição. Nota-se também, que qualquer um desses quatro estados pode ser transformados em qualquer outro, por uma transformação linear inversível em um dos mobits. Por exemplo, dado os operadores G e K, tal que:

$$\begin{array}{ll} G\,|0) = |1) & K\,|0) = |0) \\ G\,|1) = |0) & K\,|1) = |0) + |1) \end{array} \tag{3.15}$$

é possível produzir qualquer estado dessa base utilizando os operadores G e K, por exemplo:

$$|S) = G_1\,|R) \qquad |U) = K_1\,|R) \qquad |V) = K_1 G_1\,|R)\,. \tag{3.16}$$

Suponha que Alice deseja enviar uma mensagem para Bob transferindo a mensagem por um único mobit para ele. Considerando a computação clássica, ela pode, com segurança, enviar um bit (duas possíveis mensagens). Para tanto, Alice codifica a mensagem em dois estados bases $|0)$ e $|1)$ que Bob pode distinguir utilizando o observável de medição Z. Alice não pode enviar mais informações na transmissão sem a possibilidade de erro. Desta maneira, existem apenas três estados distintos do mobit válidos para Bob usar.

Mas, suponha que agora Alice e Bob inicialmente compartilham um par de mobits em um estado conjunto $|R)$. Alice pode codificar dois bits (quatro possíveis mensagens), escolhendo por aplicar operadores G, K, ou KG para seu mobit, resultando em um dos quatro estados $|R)$, $|S)$, $|U)$ e $|V)$. Se Alice entrega seu mobit transformado para Bob, ele pode realizar uma medição conjunta em ambos mobits, para com segurança distinguir a possibilidade.

3.7 Algoritmo da Satisfação Booleana UNIQUE-SAT

O algoritmo UNIQUE-SAT (algoritmo da satisfação booleana) é um problema da classe NP-Completo. Dada uma fórmula booleana, escrita somente com variáveis, parênteses e os operadores lógicos OR, AND e NOT, o objetivo do algoritmo é determinar se existe uma determinada valoração para as variáveis da fórmula que torne o resultado como *verdadeiro*. Caso a fórmula em questão seja avaliada como *verdadeiro* então ela é considerável *satisfazível* caso contrário *insatisfazível*.

Em um trabalho recente desenvolvido por Willcock e Sabry (2011), verificou-se a possibilidade de resolver esse problema utilizando a Teoria Quântica Modal. A resolução proposta

é a seguinte. Dada uma função $f : Bool^n \rightarrow Bool$, constrói-se uma ***Black Box*** U_f da seguinte maneira:

$$U_f \left|y\right\rangle \left|\bar{x}\right\rangle = \left|y \veebar f(\bar{x})\right\rangle \left|\bar{x}\right\rangle \tag{3.17}$$

A notação $\bar{x}$ representa a sequência $x_1, x_2, x_3 ... x_n$ de n bits. Os operadores s e $s^\dagger$ realizam as seguintes alterações no estado:

$$\begin{array}{ll} s\left|0\right\rangle = \left|\sigma\right\rangle & s^\dagger\left|0\right\rangle = \left|0\right\rangle \\ s\left|1\right\rangle = \left|1\right\rangle & s^\dagger\left|1\right\rangle = \left|\sigma\right\rangle \\ s\left|\sigma\right\rangle = \left|0\right\rangle & s^\dagger\left|\sigma\right\rangle = \left|1\right\rangle \end{array} \tag{3.18}$$

O algoritmo é representado pela Figura 3.1, e consiste nos seguintes passos:

1. Inicializa-se $n+1$ qubit no estado $\left|0\right\rangle \left|0\right\rangle \cdots$.
2. Aplica-se o mapeamento s para cada qubit no segundo componente do estado.
3. Aplica-se a *black box* U_f na entrada do estado.
4. Aplica-se o mapeamento s para cada qubit no segundo componente do estado.
5. Aplica-se o mapeamento $s^\dagger$ no primeiro componente do estado.
6. Deixe o primeiro componente do estado ser a. Aplica-se o mapeamento X_a para cada qubit no segundo componente do estado.
7. Aplica-se o mapeamento $s^\dagger$ no primeiro componente do estado.
8. Mede-se o estado resultante da base padrão para $n+1$ qubits. Se a medição e $\left|0\right\rangle \left|\bar{0}\right\rangle$ então a função f é **insatisfazível**. Se a medição é qualquer outro resultado a função f é **satisfazível**.

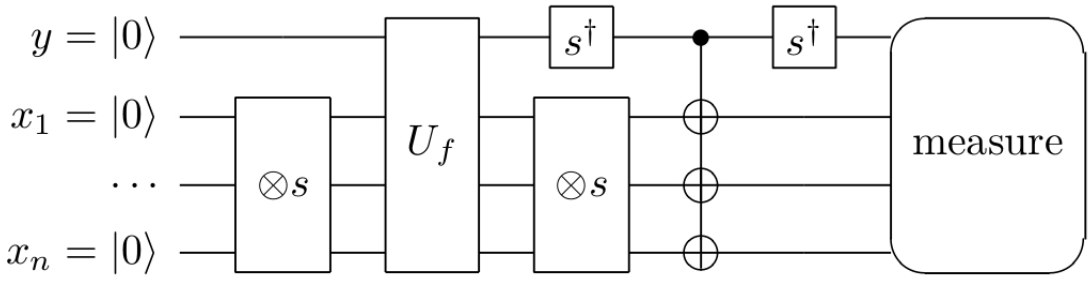

Figura 3.1 – Circuito quântico do algoritmo UNIQUE-SAT (WILLCOCK; SABRY, 2011).

3.8 Resumo do Capítulo

Neste Capítulo investiga-se uma visão essencial sobre a Computação Quântica Modal, esta teoria que serve como base para o modelo de computação desenvolvido neste trabalho. Apresenta-se as características que formam um campo finito e a descrição formal do modelo quântico modal. Além disso, pode-se ressaltar que a utilização estruturas matemáticas discretas não são uma novidade no meio científico da computação quântica, ou seja, esta possibilidade ja foi explorada em outros trabalhos, mas cada um com a sua devida particularidade. Por fim, apresenta-se a resolução teórica do algoritmo da codificação superdensa e o algoritmo da satisfação booleana UNIQUE-SAT considerando a TQM.

4 PROGRAMAÇÃO FUNCIONAL: HASKELL

Haskell é uma linguagem de programação puramente funcional de propósito geral. Toda a sua programação é realizada através de definições e chamadas de funções. O seu sistema de tipos é estático e implícito sendo que a verificação dos tipos é realizada pelo compilador, embora não haja necessidade de declará-los explicitamente. A linguagem é *lazy*, ou seja, a avaliação de expressões é realizada apenas se necessário. Essa característica permite, por exemplo, o uso de estruturas de dados infinitas, pois considera apenas a quantidade finita necessária para a validação da expressão global. Além disso, pode economizar tempo não avaliando sub-expressões desnecessárias (PEYTON-JONES, 2003).

Linguagens funcionais, possuem nativamente suporte a funções recursivas, funções de alta ordem, tipos de dados algébricos definidos pelo usuário, casamento de padrões, compreensão de listas, sistema monádico de *I/O*, dentre outros. Essa combinação de características, possibilita que a construção de funções seja realizada com uma grande abstração e síntese.

Um exemplo clássico da legibilidade e elegância de funções em Haskell é a definição da função fatorial. O fatorial de um número n, é definido pelo produto de todos os inteiros positivos menores ou iguais a n. Em Haskell pode-se facilmente modelar uma função fatorial utilizando recursão para a resolução do problema.

```
fatorial :: Int -> Int
fatorial 0 = 1
fatorial n = n * fatorial (n-1)
```

Além disso, Haskell faz uso da noção categórica de mônadas para lidar explicitamente com efeitos colaterais. Essa característica é uma das principais razões para a sua utilização como linguagem na simulação da computação quântica, pois permite modelar os efeitos computacionais quânticos separadamente a partir da estrutura monádica (MOGGI, 1989).

Na prática, devido às características citadas, Haskell possibilita realizar uma conexão com à computação quântica, principalmente por se tratar de uma linguagem que está fundamentada a partir de conceitos matemáticos e possuir um paralelismo na sua concepção, embora esse seja diferente do paralelismo encontrado na computação quântica (SABRY, 2003).

Neste capítulo investiga-se os conceitos e recursos da linguagem Haskell. A organização do capítulo está disposto da seguinte maneira. Na seção 4.1, descreve-se os tipos de dados e construções em Haskell como: classes de tipos, instancias de tipos e tipos sinônimos. A seção 4.2 apresenta o conceito de recursão em linguagens funcionais. A seção 4.3 apresenta listas e

tuplas que são fundamentais para a implementação das estruturas para a computação quântica no modelo proposto. A seção 4.4 apresenta o conceito de mônadas. Por fim, na seção 4.5 apresenta-se uma sumarização do capítulo e as principais bibliografias.

4.1 Tipos de Dados e Construções em Haskell

Haskell possui um sistema de tipos conhecido como *fortemente tipada*. Nesse caso, toda função variável ou constante possui apenas um tipo que sempre pode ser determinado. Embora, seja uma linguagem fortemente tipada ela possui um sistema de dedução automática de tipos para as funções cujos tipos não necessitam ser explicitamente determinados.

Toda definição de funções possui uma prototipação de tipos, ou seja, segue uma sequência de argumentos, sendo que o último valor à direita é o retorno da função, seguindo a seguinte estrutura:

```
function :: Type_arg → Type_arg ... Type_arg_out
```

Pode-se interpretar a sintaxe da notação de uma função da seguinte forma, o caractere "::" denota *do tipo*, enquanto que os tipos explícitos sempre são referidos pela sua letra maiúscula. Além disso, ao declarar uma função pode-se escolher entre declarar seus tipos explicitamente ou não.

Quando os tipos não são declarados, cria-se funções com tipos variáveis, isso basicamente significa que uma função pode receber ou ter como saída qualquer tipo. Esse mecanismo permite a escrita de funções com atribuições mais genéricas, ou seja, possibilita que o processamento seja eficiente para diferentes tipos. Funções cujos os tipos são variáveis são denominados funções *polimórficas*. Para exemplificar, considere a função *head*:

```
head :: [a] → a
```

A função *head* é uma função nativa de Haskell quem tem como objetivo receber uma lista e retornar o primeiro elemento. Note, que devido a definição da função ser genérica, pode-se utilizar esta função de forma global independente do tipo contido na lista.

4.1.1 Classes, Instância e Sinônimos de Tipos

De modo geral, classes de tipo são utilizadas em Haskell com o propósito de definir funções genéricas. Como visto, as funções em Haskell são de um determinado tipo, embora seja possível utilizar polimorfismo, de fato, não importa qual é o tipo da função, desde que haja uma função específica que possa trabalhar com o tipo definido.

Pode-se fazer uma analogia de classe de tipos (*Typeclass*) como uma interface em Java que define um comportamento. Por exemplo, se um tipo é parte de uma *typeclass* significa que ele suporta e implementa o comportamento específico para aquela classe de tipo.

Utiliza-se classe de tipos no modelo proposto para a computação quântica modal com o propósito de criar uma base de valores observáveis clássicos. O construtor $basis :: [a]$ listas os elementos da base. A implementação da classe é descrita abaixo e utiliza classes derivadas que além de herdar os comportamentos da classe Eq também implementa da classe *Basis*

```
class Eq a ⇒ Basis a where
  basis :: [a].
```

O símbolo "=>" é denominado como restrição de classe (*class constraint*). Pode-se interpretar essa declaração da seguinte forma, o observável a herda o comportamento da classe Eq e o construtor $basis :: a$ determina à estrutura do tipo, para este caso uma lista de booleanos.

```
instance Basis Bool where
  basis = [False, True]
```

Uma característica de suma importância é que Haskell permite a criação de novos tipos de dados de forma simples. Realiza-se a construção de novos tipos através da palavra-chave *data*. Por exemplo, considere o tipo Fp que é relativo à classe de Campos Finitos.

```
data Fp = F Integer Integer deriving (Show, Eq, Ord)
```

O tipo Fp possui dois valores inteiros que são respectivamente relativos ao valor da ordem e elemento do campo finito, enquanto a letra F representa o nome da estrutura. Construtores de valores (*value construtors*) representam basicamente qual os valores que o novo tipo pode assumir, neste caso para ambos o tipo é *Integer*. A leitura dessa construção de tipo é realizada da seguinte forma: O tipo *Fp* deve receber dois valores *Integer* e herdam as características relativas às classes *Show, Eq, Ord*.

Semelhantemente, Tipos Sinônimos são uma maneira de dar nomes que sejam mais legíveis ao contexto de programação. Ou seja, basicamente a declaração *type* permite associar um novo identificador a um tipo que já existe. Para a instanciação de novos tipos, utiliza-se a palavra reservada *type*. Ressalta-se que com a criação do novo tipo não é exatamente um novo tipo, mas sim um *sinônimo* com o nome mais adequado.

O que diferencia as funções dos novos tipos é a letra maiúscula no início de cada tipo. Por exemplo, para criar um tipo Mqbit (modal qubit) em Haskell, utiliza-se tipos sinônimos, de maneira que o código seja:

```
type Mqbit a = a → Fp
```

Neste exemplo, a *Mqbit* é uma função que realiza um mapeamento de um vetor a para um campo finito FP. O vetor representado por a faz parte da $basis$ denotada acima.

4.2 Recursão

A recursão é de suma importância para Haskell, uma vez que, ela possibilita criar soluções concisas e elegantes para problemas pensando de forma recursiva. Deve-se lembrar, que diferente de outras linguagens, em Haskell não existem laços de repetições, por isso a recursividade é utilizado em diversas ocasiões para a resolução de problemas que necessitam de repetições.

Basicamente pode-se definir a recursividade como a possibilidade de tornar uma função capaz de invocar a si própria. Comumente, recursão é utilizada para expressar definições matemáticas, logo, a sequência de Fibonacci pode ser utilizada como um bom exemplo para exemplificar o funcionamento da recursão.

A sequência de Fibonacci consiste em uma sucessão de números, tais que, definindo os dois primeiros números da sequência como 0 e 1, os números seguintes serão obtidos por meio da soma dos seus dois antecessores. O conjunto da sequência de Fibonacci, são os números: {0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,...}. Matematicamente ela é expressa pela seguinte fórmula.

$$fib(n) = \begin{cases} 0\,, & \text{se } n = 0\,; \\ 1, & \text{se } n = 1; \\ fib(n-1) + fib(n-2) & \text{demais casos.} \end{cases} \tag{4.1}$$

A abstração da função matemática da sequência de Fibonacci para Haskell pode ser definida pela função *Fib*, e tem duas particularidade que deve-se ressaltar. Primeiro, é necessário definir o caso de parada, isso é fundamental para que o programa não entre em *loop* infinito,

nesse caso especificamente existem duas definições para os casos base $fib\ 0 = 0$ e $fib\ 1 = 1$, ou seja, para os casos que a função receber 0 e 1 ira retornar valores definidos. Por fim, para os demais casos a função ocorrerá recursivamente até chegar justamente na ordem de parada ou **caso base**.

```
fib :: Int → Int
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib(n-2).
```

4.3 Listas e Tuplas

Tipos básicos podem ser combinados facilmente de duas formas: listas, que estão dispostas entre [colchetes] e tuplas entre (parênteses). Listas são estruturas de dados muito úteis em Haskell elas podem modelar dados de diversas formas, com uma ressalva, listas não possuem suporte a tipos de dados diferentes, ou seja, não é possível criar uma lista com inteiros e caracteres.

Tuplas, por sua vez, possuem suporte a tipos distintos, possibilitando criar estruturas heterogêneas. Entretanto, existe a possibilidade de utilizar as duas estruturas de listas e tuplas simultaneamente, por exemplo uma lista de tuplas que permite descrever um mobit. Para esse exemplo note que os valores a são elementos classe $basis$, logo *True* ou *False* associado a um campo finito descrito pela notação F 2 n.

```
mobit_0 = [(False, F 2 1), (True, F 2 0)].
```

Destaca-se que o mapeamento entre os elementos da base (vetores) e campos finitos é realizado através de uma camada monádica que será apresentada na próxima seção.

4.4 Mônadas em Haskell

Linguagens puras podem ser consideradas mais simples, pois, permitem que raciocínio algébrico existente na matemática, seja aplicado na resolução de algoritmos sem maiores problemas. Esse fator é fundamental para que facilite a prova de corretude em seus programas(MOGGI, 1989).

A utilização de mônadas na computação, mais precisamente em linguagens funcionais foi realizada por Moggi (1989), onde o autor apresenta uma *semântica de computação de categoria-teórica*, com o objetivo de fornecer uma prova de equivalência entre programas.

Para isso ele utiliza a categoria Kleisi para mônadas. Um exemplo utilizado por ele é várias noções de computações no λ-cálculo computacional. Pode-se citar como exemplo: computações com efeitos colaterais, exceções, não-determinismo, dentre outros.

Portanto, mônada pode ser considerada uma generalização da estrutura algébrica chamada de *monoid*, utilizada na Teoria das Categorias. Sendo que, a transposição para a programação funcional foi desenvolvida por Wadler (1995), baseada fundamentalmente no trabalho de Moggi (1989) para definir as semânticas de programas. Basicamente as mônadas possibilitam discretizar os passos de um programa, lidando com transformações de estados, exceções e continuações. Sendo que Wadler conseguiu contextualizar as mônadas para serem empregadas diretamente na programação funcional.

Os principais motivos de utilizar mônadas para estruturar um programa escrito em uma linguagem puramente funcional são: a tentativa de eliminar ou minimizar problemas quando ocorrem os chamados *efeitos colaterais*, e o segundo refere-se ao fato de que a resolução do problema não quebre o paradigma de programação funcional, ou seja, que ele se mantenha puro (HEUNEN; JACOBS, 2006).

4.4.1 Monoids

Monoid pode ser definido como um conjunto $\mathbb{S}$ juntamente com uma operação binária associativa ($\bullet$) que atua sobre os membros de um dado conjunto. A operação deve ser necessariamente sobre um conjunto fechado e deve haver um membro do conjunto que é um elemento identidade para a operação binária. Um exemplo simples são os números naturais $\mathbb{N}$, juntamente com a operação de adição (+) que tem como elemento identidade zero e a operação de multiplicação ($\cdot$) com o elemento de identidade um.

A definição de *monoid* em Haskell ocorre de maneira similar. Um *monoid* é um tipo de dados (a), juntamente com uma operação binária atuando sobre o tipo de dados ($f :: a \rightarrow a \rightarrow a$) e um elemento do tipo ($id :: a$) que é o identidade da operação de binária. A função f é comumente chamada $mappend$, e o elemento de identidade é conhecido como $mempty$.

Traduzindo para uma *type-class* obtêm-se:

```
class Monoid a where
mempty :: a
mappend :: a → a → a
```

4.4.2 Mônadas

Como visto, Haskell lida com efeitos colaterais através de mônadas, fornecendo ao programador a possibilidade de construir computações usando os blocos sequenciais. Assim, mônadas determinam como as computações combinadas formam uma nova computação dando ao programador a capacidade de definir manualmente cada vez que necessitar. Resumidamente, pode-se pensar mônada como uma estratégia para combinar computações dentro de computações mais complexas (MOGGI, 1989).

A definição de uma mônada é através de classes de tipos, dada pelo construtor m e mais duas funções monádicas conhecidas como `return` e »= (*bind*). A função *return* constrói valores do tipo `m` enquanto que a função »= (*bind*) combina as duas computações sucessivas, definindo uma computação composta. Embora recomendadas, tais funções não são rigorosamente necessárias para a criação de monádicas.

A sustentação de mônadas está na instância da classe *Monad*, propiciando o uso da característica dessa classe para escrever um código elegante. Dessa maneira, para definir uma estrutura monádica é necessário:

- um tipo construtor **m**;
- ima função *return*.
- uma função *bind* (»=).

Segundo Green (2010), mônadas em Haskell pode ser pensado como um construtor de tipo cujos membros descrevem o comportamento monádico disponível. A estrutura monádica mais simples é a função *Maybe*. Para facilitar a compreensão de mônadas pode-se primeiramente entender como é a construção da função *Maybe*.

```
data Maybe a = Nothing
             | = Just a
```

Um membro do tipo *Maybe* é *Just a* ou *Nothing* que é o construtor utilizado para descrever uma falha na computação. Considere uma operação de divisão de inteiros por exemplo, se for utilizado o divisor zero acarretará em um erro na computação completa. Um tipo monádico, por outro lado, realiza uma operação de ligação para que esse resultado não interfira com o resto da computação.

```
instance Monad Maybe where
return              = Just
Nothing     >>= f   = Nothing
(Just x )   >>= f   = f x
```

Agora, um valor *Nothing* é enviado através da extremidade da computação, mas se um valor *Just* for encontrado, então é utilizado de forma normal para o valor do tipo subjacente (anterior).

4.5 Resumo do Capítulo

Haskell fornece uma facilidade de criar novos tipos de dados, mapear valores, lidar com listas e tuplas alem de possuir uma proximidade matemática por ter como base λ-cálculo, além de apresentar uma espécie de paralelismo nativo. Todas essas características, motivam a ideia da construção de um modelo para a simulação de computação quântica seja desenvolvida em Haskell. Listas e tuplas são fundamentais para a modelagem de qubtis, estados emaranhados e estruturas quânticas. Entretanto, a forma com que Haskell lida com efeitos colaterais é, sem dúvida, a principal característica para modelagem de efeitos quânticos na linguagem. Para aprofundar os conhecimentos em Haskell, uma boa fonte é a documentação e tutoriais dispostos em *www.haskell.org* além dos trabalhos de Wadler (1995) e Moggi (1989).

5 MODELOS MONÁDICOS PARA COMPUTAÇÃO QUÂNTICA EM HASKELL

No que se diz respeito a linguagens funcionais puras como Haskell, não é possível resolver problemas não-determinísticos, pois quebra uma das suas regras, a *transparência referencial*. Esta regra define que, dada uma função que tem como entrada um valor, para o mesmo valor sempre deverá gerar o mesmo resultado (WADLER, 1995).

Porém, segundo Mu e Bird (2001), bits quânticos podem ser pensados como um tipo de efeito computacional, mais especificamente como uma computação não-determinística. Logo, para não quebrar a regra de transparência referencial, é necessária a utilização de mônadas para modelar esses tipos de efeitos computacionais.

Neste capítulo, apresenta-se um estudo relacionado aos modelos de computação quântica que utilizam como linguagem básica Haskell e aplicam mônadas como característica fundamental para a estruturação de efeitos quânticos. A organização deste Capítulo está disposta da seguinte forma: A seção 5.1 apresenta um estudo referente ao trabalho *Structuring Quantum Effects: Supeoperator as Arrows* (VIZZOTTO; ALTENKIRCH; SABRY, 2006), que aplica uma generalização de mônadas, conhecida como *Arrows* para resolver o problema da medição quântica. Na seção 5.2 demonstra-se o trabalho de Altenkirch e Green (2009) denominado *Quantum IO Monad*, que elabora uma biblioteca para computação quântica em Haskell baseada em mônadas. Por fim, na seção 5.3 apresenta-se um breve resumo do capítulo.

5.1 Estruturando Efeitos Computacionais: Superoperadores com Arrows

A construção de efeitos computacionais, tais como: atribuição, exceções e não-determinismo são características que podem ser modeladas utilizando uma construção categórica de mônadas. Entretanto, não é possível expressar outra classe de efeitos quânticos como a medição, decoerência e ruído quântico. A medição quântica é um efeito computacional crítico para computação, que embora possa ser resolvido utilizando o *princípio da medição adiada* e o *controle clássico de efeitos colaterais*, ressalta-se que ambas as situações não apresentam boas soluções (MOGGI, 1989).

Nesse trabalho realizado por Vizzotto, Altenkirch e Sabry (2009) apresenta uma solução para a modelagem da medição quântica. A proposta é baseada em um modelo generalizado de mônadas, conhecido como *Arrows*. Com essa nova proposta, o estado da computação é repre-

sentado utilizando *matriz densidade* e as operações são realizadas através de *superoperadores*. Ressalta-se que, primeiramente é necessário investigar como é realizada a construção em modelos tradicionais da computação quântica em Haskell, ou seja, através do mapeamento de vetores e operadores lineares.

5.1.1 Vetores

Uma maneira de construir vetores em Haskell é realizar o mapeamento de um valor clássico observável para um vetor. Considere a, um elemento da base definida pelo construtor $Basis :: a$. Um valor quântico puro é um mapeamento do tipo $a \rightarrow \mathbb{C}$, que associa cada elemento da base com uma amplitude de probabilidade complexa, representada pelo conjunto $\mathbb{C}$.

```
Class Eq a ⇒ Basis a where basis :: [a]
type C = Complex Double
type vec a = a → C
```

Mônadas proporcionam construir computações em termos de valores e sequências de computações utilizando esses valores. Nesse tipo de construção, vec não é especificamente uma mônada, mas corresponde a uma estrutura Kleisli e, portanto, a probabilidade introduzida por espaços vetoriais constitui um efeito computacional que pode ser estruturado utilizando uma leve generalização de monadas em Haskell.

```
return :: Basis a ⇒ a → Vec a
return a b = if a ≡ b then 1.0 else 0.0

(>>=) :: Basis a ⇒ Vec a → (a → Vec b) → Vec b
va >>= f = λb → sum [(va a) ∗ (f a b) | a ∈ basis]
```

Embora a computação seja aplicada em uma base específica, os tipos são mais específicos do que os utilizados nativamente por mônadas, mas, isso não deixa de satisfazer as três regras de mônadas.

A partir da construção de mônadas, é possível definir produtos sobre vetores, como: produto escalar $\otimes$, produto tensorial $\oplus$ e interno $\langle\cdot\rangle$, da seguinte maneira:

```
(⊗) :: K → Vec a → Vec a
pa ⊗ v = λa → pa * v a

(⊕) :: Vec a → Vec b → Vec (a,b)
v1 ⊕ v2 = λ (a,b) → v1 a * v2 b

(⟨·⟩) :: Basis a ⇒ Vec a → Vec a → K
v1 ⟨·⟩ v2 = sum [conjugate (v1 a) * (v2 a) | a ∈ basis]
```

Uma vez definidos esses operadores, pode-se então criar exemplos de mapeamento de *booleanos* para vetores, por exemplo, estruturando os qubits: $qFalse$, $qTrue$, qFT e $qFmT$.

```
instance Basis Bool where basis = [False,True]
qFalse,qTrue :: Vec Bool
qFT, qFmT :: Vec Bool
qFalse = return False
qFT = (1/√2) ⊗ (qFalse `mplus` qTrue)
qTrue = return True
qFmT = (1/√2) ⊗ (qFalse `mminus` qTrue)
```

Note que os dois primeiros são vetores unitários correspondentes aos elementos da base e os últimos dois correspondem aos estados em superposição de $False$ e $True$. Considerando a notação de Dirac esses vetores correspondem respectivamente a: $|False\rangle$, $|True\rangle$, $\frac{1}{\sqrt{2}}(|False\rangle + |True\rangle)$, $\frac{1}{\sqrt{2}}(|False\rangle - |True\rangle)$. Vetores de diversos valores podem ser descritos através do produto tensorial ou produto cartesiano da base fundamental:

```
instance (Basis a, Basis b) ⇒ Basis(a, b) where
basis = [(a, b) | a E basis, b ∈ basis]
p1,p2,p3:: Vec (Bool,Bool)
p1 = qFT ⊕ qFalse
p2 = qFalse ⊕ qFT
p3 = qFT ⊕ qFT
epr :: Vec (Bool,Bool)
epr (False,False) = 1/√2
epr (True,True)   = 1/√2
epr_              = 0
```

No caso acima, em contraste com os primeiros três vetores, o último descreve um estado emaranhado, ou seja, que não pode ser separado do produto independente do estado quântico. O nome do vetor epr refere-se ao estado emaranhado proposto por Einsten, Podolski e Rosenberg (BELL, 1964).

5.1.2 Operadores Lineares

Dados os conjuntos A e B, um operador linear $f \in A \multimap B$ é uma função mapeando o vetor A sobre o vetor B:

```
type Lin a b = a → Vec b
```

A definição de uma operação linear especifica a ação em um elemento individual da base. Para aplicar uma operação linear f em um vetor v, utiliza-se a operação *bind* para calcular f»=v. Por exemplo, "qFT »= *hadamard*" aplica a porta *hadamard* no vetor qFT que produz como resultado o vetor $qFalse$.

Com essa característica é possível criar funções de alta-ordem, que recebem operadores lineares e tem como retorno novos operadores lineares. Isso é muito importante, pois é possível, definir *operações controladas*. Além disso, combinar com transformações de diversas formas, possibilitando criar um operador correspondente ao produto externo.

5.1.3 Matriz Densidade

Uma matriz densidade pode ser pensada como uma perspectiva estatística do estado do vetor. No formalismo da matriz densidade, um estado quântico que é utilizado para descrever um vetor v, agora é modelado pelo seu produto externo.

```
type Dens a = Vec (a,a)

pureD :: Basis a ⇒ Vec a → Dens a
pureD v = lin2vec (v ⟩*⟨ v)

lin2vec :: (a → Vec b) → Vec (a,b)
lin2vec = uncurry
```

A função $pureD$ envolve um estado na representação da matriz densidade. Por conveniência, utiliza-se a função *uncurry* para os argumentos da matriz densidade ficar mais próxima à notação de uma matriz. Por exemplo, as matrizes densidade que correspondem aos vetores $qFalse$, $qTrue$ e qFT podem ser representadas respectivamente por:

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \quad \text{e} \quad \begin{pmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{pmatrix}. \tag{5.1}$$

De fato, as matrizes densidade podem representar outros estados além de estados puros. Especificamente, se realizada a medição no estado representado por qFT, deve-se obter $False$ com probabilidade $1/2$ ou $True$ com $1/2$. Essa informação que não pode ser expressa utilizando vetores, e por outro lado, pode ser representada pela seguinte matriz densidade.

$$\begin{pmatrix} 1/2 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 1/2 \end{pmatrix} = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix}. \tag{5.2}$$

Essa matriz densidade representa um "estado misto" que corresponde a soma (normalizada) da matriz densidade para os dois resultados da observação.

5.1.4 Superoperadores e Medição Quântica

Operadores mapeando matrizes densidades para matrizes densidades são chamados de superoperadores. A definição em Haskell é a seguinte:

```
type Super a b = (a,a) → Dens b
lin2super :: (Basis a, Basis b) => Lin a b → Super a b
lin2super f (a1,a2) = lin2vec (f a1 ⟩∗⟨ f a2),
```

sendo que, a função $lin2super$ constrói um superoperador de um operador linear em vetores.

A grande diferença para o outro modelo de computação quântica, é que utilizando superoperadores e matrizes densidades é possível modelar a função $meas$ (medida quântica) para que possua ambos resultados, ou seja, o valor clássico (colapsado) e o estado quântico do sistema pós-medição.

```
trL :: (Basis a, Basis b) ⇒ Super (a,b) b
trL ((a1,b1),(a2,b2)) = if a1 = a2 then return (b1,b2) else mzero

meas :: Basis a ⇒ Super a (a,a)
meas (a1,a2) = if a1 = a2 then return ((a1,a1),(a1,a1)) else mzero
```

Note que a função trL expressa acima, tem como papel "esquecer" o valor colapsado, considerando a função $pureDqFT >>= meas >>= trL$, então o resultado terá retorno o estado pré-medição:

$$\begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix}. \tag{5.3}$$

5.1.5 Superoperador com Arrows

Arrows permite que a aplicação de um superoperador em uma matriz densidade seja realizada através de uma operação monádica, da seguinte forma:

```
(>>=) :: Dens a → ((a,a) → Dens b) → Dens b
```

Porém, agora a computação "consome" múltiplos valores de entrada. De fato, a adição para definição da noção do procedimento que pode realizar efeitos computacionais, *arrows* pode ter um componente estatístico independente da entrada, ou ainda aceitar mais que uma entrada.

Em Haskell, uma interface *Arrow* é definida usando a seguinte classe de declaração:

```
class Arrow a where
arr :: (b → c) → a b c
(>>>) :: a b c → a c d → a b d
first :: a b c → a (b,d) (c,d)
```

Para ser uma *arrow*, um tipo "a" deve dar suporte a três operações arr, $>>>$ e $first$, com dado tipo de dados. As operações devem satisfazer as seguinte equações:

```
arr id >>> f = f
f >>> arr id = f
(f >>> g) >>> h = f >>> (g >>> h)
arr (g.f) = arr f >>> arr g
first (arr f) = arr (f × id)
first (f >>> g) = first f >>> first g
first f >>> arr (id × g) = (id × g) >>> first f
first f >>> arr fst = arr fst >>> f
first (first f) >>> arr assoc = arr assoc  >>> first f
```

As funções $assoc$ e $\times$ são definidas como segue:

```
(f × g) (a, b) = (f a, g b)
assoc ((a, b), c) = (a, (b, c))
```

A função arr permite introduzir *arrows* "puras", que são funções de suas entradas para suas saídas. A função $>>>$ é similar a »=, composta de duas computações. A função *first*, permite-nos aplicar uma *arrow* a um componente do estado global. As equações acima garantem que essas operações sempre sejam bem-definidas, mesmo com permutações arbitrárias e troca de associatividade.

5.2 The Quantum IO Monad

Nesta seção apresenta-se um estudo relacionado ao trabalho de Altenkirch e Green (2009) denominado de *The Quantum IO Monad*. Ele apresenta uma interface escrita em Haskell para a computação quântica disponível como biblioteca, denominada *QIO Monad*. Ressalta-se que o objetivo principal é apresentar a maneira com que o trabalho utilizou para lidar com os efeitos quânticos utilizando mônadas, deixando de lado as particularidades propostas para a interface.

A biblioteca QIO, separa a parte computacional reversível (unitária) e irreversível (probabilística) e fornece uma operação reversível, permitindo o uso de um qubit auxiliar (*ancilla*) em um design modular. Programas QIO podem simular as amplitudes de probabilidades calculando a probabilidade distributiva ou incorporando na mônada IO por meio de um gerador de números randômicos.

A investigação tem como objetivo a implementação de uma interface monádica para a programação quântica em Haskell. As principais vantagens apresentadas pelo modelo são:

- Explorar os meios existentes de abstração presentes em Haskell para estruturar os programas quânticos. Um exemplo disso é representando a implementação da classe *Qdata* que relaciona tipos de dados clássicos com a contraparte quântica;
- Possibilidade de verificar que os primitivos semânticos são mais úteis à estrutura programas quânticos, sem se comprometer com uma visão particular precipitada;
- Utilizar o suporte existente para Haskell, na forma de ferramentas e bibliotecas para desenvolver os programas quânticos.

5.2.1 Mônadas QIO

No modelo proposto, um vetor sobre os tipos x e a é definido como uma lista de pares desses tipos.

```
data Vec x a = Vec{unVec :: [(a, x)]}
```

A definição de funções sobre vetores são:

```
empty = Vec x a
empty = Vec []

(@) :: (Num x , Eq a) ⇒ Vec x a → a → x
(Vec ms) @ a = foldr (λ(b, k) m → if a ≡ b then m + k else m) 0 ms

(⊗) :: Num x → x → (Vec x a) → Vec x a
l ⊗ (Vec as) = (Vec (map (\(a, k ) → (a, l ∗ k )) as))

(⊕) :: (Vec x a) → (Vec x a) → Vec x a
(Vec as) ⊕ (Vec bs) = (Vec (as ++ bs)).
```

A função *empty* corresponde a um vetor vazio. A função de pesquisa @ requer que o tipo de x seja da classe *Num* (numérico) e o tipo a da classe *Eq*, e retorna a soma do argumento numérico de todos os membros do vetor que corresponde a determinado argumento. A função $\otimes$ é a multiplicação por escalar, que recebe um x do tipo numérico e pode multiplicar a parte numérica de cada vetor por um escalar. A função $\oplus$ representa a união de vetores, ou seja, ela une dois vetores utilizando a concatenação de listas.

Após determinada todas as funções é então possível definir vetores sobre tipos numéricos como uma mônada especificamente, de tal forma que eles possam ser usados para manter as distribuições de probabilidades necessárias.

A função de *return* leva um valor base para o tipo mônada apenas dando-lhe uma probabilidade de 1 (um) como o único membro da distribuição de probabilidade. A operação de ligação (*bind*), aplica a função associada ao primeiro argumento de cada membro distribuição de probabilidade atual, e multiplica o argumento numérico por este novo resultado, com o efeito de atualizar a distribuição de probabilidade de acordo com o efeito da dada função.

```
instance Num n → Monad (Vec n) where
return a = Vec [(a, 1)]
(Vec ms) >>= f = Vec [(b, i ∗ j ) | (a, i ) <− ms, (b,j) <− unVec (f a)]
```

Para fazer uso desses vetores na implementação de QIO, define-se uma *PMonad* (mônada probabilidade) como uma mônada, juntamente com a função extra *merge*. A função *merge*, pode ser pensada como a função que define como as computações (c,d) do mesmo tipo podem ser misturadas dependendo do argumento $\mathbb{R}$, onde $merge\ p\ c\ d$ significa que a probabilidade de c ocorrer é p, e a probabilidade de d ocorrer é $1 - p$.

```
class Monad m ⇒ PMonad m where
merge :: ℝ → m a → m a → m a
```

Para criar a distribuição de probabilidade para a função *sim* é necessário definir o tipo atual da distribuição de probabilidade *(Prob a)*.

```
data Prob a = Prob {unProb :: Vec R a}
```

Nesse caso a distribuição pode ser sobre qualquer tipo a com um argumento $\mathbb{R}$ com a sua probabilidade atual. A definição a probabilidade *Prob* como uma mônada levando as operações monádicas para *Vec*.

```
instance Monad Prob where
return = Prob ∘ return
(Prob ps) >>= f = Prob (ps >>= unProb ∘ f )
```

```
instance PMonad Prob where
merge pr (Prob ift) (Prob iff) = Prob ((pr⊗ift) ⊕ ((1−pr)⊗iff))
```

A função *PMonad* é então definida com a função *merge* multiplicando cada distribuição de probabilidade por a respectiva probabilidade e associando elas. Em comparação a função *run* requer que "IO Monad" seja definido como um PMonad, isso é possível utilizando um gerador de números aleatórios para retornar probabilisticamente um dos argumentos determinados dependendo da probabilidade dada:

```
instance PMonad IO where
merge pr ift iff = Random.randomRIO (0, 1.0)
>>= λ pp → if pr > pp then ift else iff
```

5.3 Resumo

Neste capítulo, apresenta-se um estudo referente aos trabalhos de Vizzotto, Altenkirch e Sabry (2006) e Altenkirch e Green (2009). Respectivamente, o primeiro trabalho elabora uma solução para lidar com efeitos quânticos, mais especificamente com a *medição quântica*, cujo efeito não foi contemplado pela investigação que considera operadores lineares e mapeamento de vetores para escalares realizada por Mu e Bird (2001). A solução proposta pelo trabalho é através de uma generalização de mônadas, conhecida como *arrows*, utilizando superoperadores e matriz densidade. No segundo trabalho, é construída uma interface disposta como uma biblioteca em Haskell, a ideia é implementar os efeitos quânticos utilizando mônadas, com o objetivo de facilitar o desenvolvimento de algoritmos quânticos trazendo uma semântica formal para o programador.

6 IMPLEMENTAÇÃO DO MODELO QUÂNTICO MODAL EM HASKELL

A complexidade da computação quântica requer esforços que não são totalmente contemplados em linguagens clássicas, sejam elas dos mais diversos tipos de paradigmas: estruturados, funcionais, lógicos, entre outros (SOFGE, 2008). Entretanto, investigou-se que é possível realizar a simulação de algoritmos quânticos de maneira satisfatória utilizando a computação clássica.

Neste trabalho, utiliza-se como ferramenta a linguagem Haskell para desenvolver a biblioteca MQS (*Modal Quantum Simulation*) para Computação Quântica. Este modelo, que embora bastante simplificado, se comportou eficiente para a resolução dos algoritmos da Codificação Superdensa, Teleportação Quântica e Unique-Sat.

Ressalta-se que o foco principal do trabalho além da implementação dos algoritmos da forma prática, é desenvolver uma estrutura capaz de realizar a simulação da computação quântica, propiciando um modelo versátil que possibilite a confecção de novos algoritmos com uma carga maior de complexidade no futuro, com a expectativa de que este modelo alcance um bom desempenho sem que haja uma demanda desproporcional de recursos computacionais.

Este capítulo está estruturado da seguinte maneira. Na Seção 6.1 descreve-se a tradução da estrutura de campos finitos para Haskell. Na Seção 6.2 retrata-se a utilização de mônadas associadas com a programação funcional na computação quântica. A Seção 6.3 apresenta as bases computacionais e os sistemas quânticos básicos mobit (modal qubit). A Seção 6.4 refere-se a operadores lineares, responsáveis por realizar alterações no estado de sistemas quânticos reversíveis. A Seção 6.5 descreve a operação irreversível da medição quântica. Nas Seções 6.6, 6.7 e 3.1 apresenta-se respectivamente o desenvolvimento dos algoritmos quânticos da Codificação Superdensa e da Teleportação Quântica e Unique-Sat. Por fim, na Seção 6.9 apresenta-se a sumarização geral do Capítulo.

6.1 Campos Finitos em Haskell

A utilização da estrutura matemática de campos finitos é a principal característica da computação quântica modal. Existem diversos tipos de campos finitos, mas o campo utilizado para a construção do modelo é o campo de ordem prima binária. Por padrão, todo campo finito possui uma *ordem* e seus *elementos*, que respectivamente correspondem ao tamanho e aos itens

do campo. Normalmente o método utilizado para determinar a ordem do campo é a partir da operação de módulo.

Por exemplo, ao efetuar uma operação aritmética, utiliza-se o módulo p, onde p é um número primo e para cada diferente primo p obtém-se um campo Fp que consiste no conjunto $\{0, 1, \ldots p-1\}$, com as operações matemáticas realizadas considerando o módulo p. Note que para o campo $F\ 5$, ao realizar a multiplicação $3*3 = 4$, enquanto que no campo $F\ 7$ é $3*3 = 2$.

Para realizar a abstração dessa estrutura para Haskell cria-se um novo tipo de dados, denominado FP utilizando o construtor *data*, e uma instância de tipos, ela irá determinar o comportamento do novo tipo de dado que esta sendo construído. Define-se a classe *Finite Field* da seguinte maneira:

```
data Fp = F Integer Integer deriving (Show, Eq, Ord)
```

O primeiro número inteiro p representa a ordem do campo finito, enquanto que o segundo a representação do valor. Portanto, a sintaxe definida para determinar um campo finito é realizada da seguinte forma: $F\ P\ 2 = \{F\ 2\ 0, F\ 2\ 1\}$, onde respectivamente, F significa a classe *Finite Field*, 2 representa a ordem e por último os valores 0 ou 1 retratam os elementos pertencentes ao campo definido.

Para determinar as operações matemáticas de soma e multiplicação, estabelece-se o comportamento da classe Fp a partir de uma instância *Num*. Ou seja, ela herda as características numéricas. A primeira necessidade é verificar se o campo é de mesma ordem (logo $p == q$) deve ser verdadeiro, portanto, para realizar a operação desejada dos valores deve ser considerado a aritmética modular binária.

```
instance Num Fp where
   F p x + F q y | p == q = F p $ (x+y) `mod` p
   F p x * F q y | p == q = F p $ (x*y) `mod` p
   negate (F p x) = F p (-1*x)
   abs (F p x) = if (x < 0 ) then (F p ((-1)*x)) else (F p x)
```

Um dos problemas mais comuns encontrados na utilização de campos finitos é garantir que as operações sejam realizadas por campos de mesma ordem, por exemplo, uma operação matemática entre os campos $F\ 2$ (binário) e $F\ 3$ (terciário) não deve ser possível. Contudo, no modelo proposto esse não é problema, uma vez que utiliza-se apenas campos binários.

```
instance Fractional Fp where
   F p x / F q y | p == q =  (F p x) * (recip (F p y))
   recip (F p x) = (F p (findInv p x 1))
```

A função $findInv$ determina o inverso de x módulo p, ela recebe o valor 1 inicialmente e incrementa até achar o inverso, por exemplo $17\ mod\ 29 = 12$, pois $17 * 12\ mod\ 29 = 1$.

```
findInv :: Integer → Integer → Integer → Integer
findInv p x y  = if (((x * y) `mod` p) == 1) then y else findInv p x (y+1)
```

6.2 Mônadas para computação quântica

Mônadas em Haskell, conceitualmente podem ser pensadas como a descrição de computações combinatórias. Na dimensão de programação, mônada é uma forma de estruturar computações em termos de valores e sequências de computação utilizando vetores e bases computacionais de campos finitos. A mônada para o modelo proposto é representada utilizando um tipo construtor m e duas funções: $mbind$ e $mreturn$.

```
mreturn :: (Basis a) ⇒ a → Mqbit a
mreturn a  b = if (a==b) then (F 2 1) else (F 2 0)

mbind :: (Basis a, Basis b) ⇒ Mqbit a → (a → Mqbit b) → Mqbit b
(ma `mbind` f) mb  = sum [(ma a) * (f a mb) | a <- basis]
```

A função $mreturn$ tem como objetivo especificar como os valores são levados para a computação, enquanto que a função $mbind$ define como ocorrerá a sequência de computações.

A função $mplus$ é responsável por criar pares de mobits, por exemplo considere o estado $|00)$, cria-se esse estado utilizando a seguinte notação $|0)$`mplus`$|0)$.

```
mplus :: (Basis a) ⇒ Mqbit a → Mqbit a → Mqbit a
mplus mq1 mq2 a = mq1 a + mq2 a
```

6.3 Bits quânticos e Bases Computacionais

Na computação quântica uma maneira de realizar a definição de um qubit em Haskell, é utilizando o mapeamento de vetores para um escalar complexo, que representa a amplitude de probabilidade (VIZZOTTO; ALTENKIRCH; SABRY, 2006). Semelhantemente, na computa-

ção quântica modal utiliza-se o mapeamento de um vetor para um campo finito para determinar o mobit.

Portanto, para a implementação do bit quântico utiliza-se o conceito de tipos sinônimos em Haskell. O construtor é a palavra *type*, para a construção de um mobit a declaração foi estruturada da seguinte forma.

```
type Mqbit a = a → Fp
```

Uma vez estabelecido o tipo do mobit, representado pelo tipo $Mqbit$ (modal qubit), é necessário definir a base computacional. Para isso, utiliza-se classe tipos de Haskell, com ela é possível determinar todas as características necessárias para essa computação. Primeiramente define-se que ela pertence a classe *Eq* e é uma lista.

```
class Eq a ⇒ Basis a where
  basis :: [a]
```

A definição das bases computacionais foram realizadas através da instância de tipos, sendo que, para definir a sua estrutura utiliza-se tuplas cobrindo todas as possibilidades da base. Dessa forma, sempre que necessário aumentar a quantidade de mobits do sistema é fundamental que a base esteja instanciada. Por exemplo, para um e dois mobits respectivamente, as bases são definidas da seguinte maneira:

```
instance Basis Bool where
  basis = [False, True]

instance Basis (Bool,Bool) where
  basis = [(False,False),(False,True),(True,False),(True,True)].
```

Se houver a necessidade de trabalhar com uma quantidade maior de mobits o requisito básico é que a instância da nova base para o sistema esteja criada. Essa é a estrutura básica para definir os mobits:

```
mobit_0 :: Mqbit Bool
mobit_0 = mreturn False

mobit_1 :: Mqbit Bool
mobit_1 = mreturn True

mobit_sig :: Mqbit Bool
mobit_sig = mobit_0 `mplus`  mobit_1
```

Por exemplo, um mobit é do tipo *Mqbit Bool* e, a função monádica *mreturn* tem o papel fundamental de fazer o mapeamento reunindo as características de campos finitos com a propriedade das bases computacionais. Com isso, obtém-se uma estrutura similar à um qubit da computação quântica convencional. Por exemplo, as informações contidas em um mobit serão as seguintes:

```
mobit_0 = [(False,F 2 1),(True,F 2 0)].
```

Esta é a representação do $mobit\_0$, onde os valores *True* e *False* que estão associados com as amplitudes de probabilidades expressas por campos finitos binários, para o valor *True* a construção é similar. O que difere é a definição do mobit_sig (em superposição), para defini-lo deve-se utilizar o operação *mplus* conforme código acima, tendo como resultado a seguinte estrutura:

```
mobit_sig = [(False,F 2 1),(True,F 2 1)].
```

Note que a operação *mplus* é monádica, logo ela tem o papel de receber dois mobits e retornar um em estado em superposição, ou seja, a criação do $mobit\_sig$, é a operação *mplus* aplicada ao $mobit\_0$ e o $mobit\_1$, nos dando como resultado o $mobit\_sig$, que é os vetores *True* e *False* com probabilidade positiva.

6.4 Operadores Lineares

Dados dois vetores A e B, um operador linear $f \in A \multimap B$ é uma função mapeando o vetor A sobre o vetor B. É possível representar tais operadores como funções que mapeiam valores para vetores, tal como foi utilizada na implementação realizada por (VIZZOTTO; ALTENKIRCH; SABRY, 2006).

Operadores lineares representam um papel fundamental no modelo proposto, pois através deles é possível realizar as transformações unitárias do sistema quântico, ou seja, aplicando um operador linear é possível modificar o estado do sistema.

```
type Lin a b = a → Mqbit b
```

Define-se o operador como um tipo *Lin* que terá o papel de mapear o valor, que neste caso será um elemento da base, ou seja, *True* ou *False* para um *Mqbit*, ou seja, mobit. Após de-

finida sua estrutura, é possível moldar alguns operadores lineares. Os operadores G e K foram utilizados por Schumacher e Westmoreland (2011) (análogos aos encontrados na computação quântica convencional); em sua teoria para realizar as transformações necessárias para desenvolver os algoritmos de codificação superdensa e teleportação em seu modelo. Consegue-se modelar similarmente em Haskell, da seguinte forma:

```
g :: Lin Bool Bool
g True  = mobit_0
g False = mobit_1

k :: Lin Bool Bool
k False = mobit_0
k True  = mobit_sig
```

Note que ambos operadores g e k recebem uma estrutura do tipo *Bool Bool*, sendo que o operador g realiza uma operação similar a de uma porta lógica clássica *not*, por exemplo quando o operador recebe um valor clássico *True* ele transforma em um valor to tipo $mobit\_0$. Enquanto que k recebe um valor *False* e não realiza nenhuma alteração e quando receber um valor do tipo *True* a operação será a transformação para um $mobit\_sig$, ou seja, mobit em superposição.

6.5 O processo da Medição Quântica Modal

O processo de medição quântica é uma tarefa mais complexa do que a existente no mundo clássico, entretanto, é de suma importância, pois precisa-se conhecer o resultado pós-processamento do algoritmo. A operação de medida no modelo proposto é uma função que terá como entrada um mobit e uma lista de mobits, representando respectivamente o mobit a ser medido e a base dual correspondente ao sistema. O resultado é uma lista de mobits que podem ser medidos em relação ao estado e à base dual. Sendo assim, a função foi estruturada da seguinte forma:

```
measure ::(Basis a) ⇒ Mqbit a → [Mqbit a] → [Mqbit a]
measure mb lmb = [mbs | mbs ← lmb, (dual mb mbs) == (F 2 1)]
```

A primeira linha representa a estrutura da função descrita acima, enquanto que a segunda é onde realmente acontece a execução da função. Como o resultado dessa função é uma lista, utiliza-se a compreensão de listas em Haskell para a produção do resultado. Note, que aparece uma nova função chamada *dual*.

```
dual :: (Basis a) ⇒ Mqbit a → Mqbit a → Fp
dual mb1 mb2 = sum [(mb1 a) ∗ (mb2 a) | a ← basis]
```

Explicou-se no Capítulo 4 como funciona o processo da medição quântica modal, contudo um dos requisitos básicos para que a medição de um estado seja possível é, que a amplitude de probabilidade resultante da operação *dual* seja igual a 1 ou *True*. Sendo assim, a função *dual* terá o papel de verificar esse requisito. Ela recebe dois mobits e, realiza a operação de multiplicação entre os mobits, com isso pode-se determinar as amplitudes de probabilidade dos elementos.

Após os resultados da função *dual* serem processados, é necessário analisá-los e, verificar quais elementos possuem a amplitude positiva. E essa é uma tarefa mais simples, pois é necessário apenas constatar se a amplitude é $\neq 0$. Finalizado esse processo, obtém-se como resultado da medição uma lista com os elementos possíveis da medição.

6.5.1 Espaço Vetorial e Base dual

A base utilizada no algoritmo da codificação superdensa e teleportação, por exemplo, é composta por quatro estados emaranhados de dois mobits $|R),|S),|U)$ e $|V)$. Para criar esses estados utiliza-se a função *mplus*, exatamente da mesma forma como foi realizado na definição do $mobit\_sig$, porém, agora levando em consideração que esta sendo trabalhado com dois mobits, logo é necessário uma base do tipo $Mqbit(Bool, Bool)$, portanto as bases são definidas da seguinte forma:

```
r :: Mqbit (Bool, Bool)
r = mreturn (False, False) `mplus` mreturn (True, True)

s :: Mqbit (Bool, Bool)
s = mreturn (False, True)  `mplus` mreturn (True, False)

u :: Mqbit (Bool, Bool)
u = mreturn (False, False) `mplus` mreturn (True, False)
        `mplus` mreturn (True, True)

v :: Mqbit (Bool, Bool)
v = mreturn (False, False) `mplus` mreturn (False, True)
        `mplus` mreturn (True, False)
```

Entretanto, é necessário associar a medição com uma base do espaço dual V^*, ou seja, toda base $|a)$ para V é associada com uma base dual $(a|$, a base dual dos quatro estados é representada abaixo:

```
r_dual :: Mqbit (Bool, Bool)
r_dual = mreturn (False, True) `mplus` mreturn (True, False) `mplus` mreturn
    (True, True)

s_dual :: Mqbit (Bool, Bool)
s_dual = mreturn (False, False) `mplus` mreturn (True, False) `mplus` mreturn
     (True, True)

u_dual :: Mqbit (Bool, Bool)
u_dual = mreturn (False, True) `mplus` mreturn (True, False)

v_dual :: Mqbit (Bool, Bool)
v_dual = mreturn (False, False) `mplus` mreturn (True, True)
```

6.6 Algoritmo da Teleportação Quântica

O algoritmo da Teleportação Quântica consiste na transmissão de um estado do mobit entre Alice e Bob sem conexão física. Primeiramente ambos compartilham um par de mobits emaranhados, porém a ideia central do algoritmo é a possibilidade de reconstruir o estado mobit sem enviar essa informação por um canal inseguro.

Suponha que Alice e Bob compartilham o estado $|R) = |00) + |11)$, para que Alice possa enviar com segurança o seu estado para Bob, ela realiza uma operação linear em seu estado, emaranhando com outro mobit neste momento gerando um estado intermediário. Para isso utiliza-se os operadores $alice_0$, $alice_1$, $alice_2$, $alice_3$, tal como no algoritmo da codificação superdensa, esses operadores tem como objetivo aplicar uma transformação unitária no primeiro mobit (mobit de Alice).

```
alice0 :: Lin (Bool, Bool) (Bool, Bool)
alice0 (top, bot) = id top `mbind` (λ a1 → mreturn (a1, bot))

alice1 :: Lin (Bool, Bool) (Bool, Bool)
alice1 (top, bot) = k top `mbind` (λ a1 → mreturn (a1, bot))

alice2 :: Lin (Bool, Bool) (Bool, Bool)
alice2 (top, bot) = g top `mbind` (λ a1 → mreturn (a1, bot))

alice3 :: Lin (Bool, Bool) (Bool, Bool)
alice3 (top, bot) = kg top `mbind` (λ a1 → mreturn (a1, bot))
```

Naturalmente, ao realizar uma transformação linear utilizando um dos operadores K_1, G_1, KG_1 dependendo do operador utilizado obtém-se como resultado um dos estados:

$$\begin{aligned} |S) &= |0,1) + |1,0)\,, \\ |U) &= |0,0) + |0,1) + |1,1)\,, \\ |V) &= |0,0) + |0,1) + |1,0)\,. \end{aligned} \tag{6.1}$$

Neste momento o sistema está em um estado intermediário, portanto, Bob não sabe que o sistema sofreu uma alteração. Agora dependendo da informação que Alice deseja transferir para Bob, ela não precisa enviar o estado completo por um canal inseguro e sim apenas uma informação clássica para ele saber o que precisa medir no sistema, dessa maneira, se a informação clássica for interceptada não irá fazer nenhum sentido para o receptor, pois ele necessitará da informação que está com Bob para realizar a medição correta do sistema completo.

Portanto, a estrutura do algoritmo da teleportação e descrita da seguinte forma:

```
telep :: Mqbit (Bool,Bool) → Int → [Mqbit (Bool,Bool)]
telep st b = case b of
             1→ measure (st `mbind` alice0) basedual
             2→ measure (st `mbind` alice1) basedual
             3→ measure (st `mbind` alice2) basedual
             4→ measure (st `mbind` alice3) basedual
```

A função $telep$ recebe o estado emaranhado e um valor inteiro representando a informação clássica e possui como retorno uma tupla de mobits que será resultado do processo de medição. A definição de qual operação deve ser realizada é expressa pela condicional *case*, desta forma, dependendo da informação clássica será realizada uma operação de medida com a devida alteração do estado inicial.

Por exemplo, se considerar como entrada para a função $telep$ o estado $|R)$ e a informação clássica 1, a transformação resultará no estado intermediário $|S)$. Portanto, quando Bob realizar a medição do estado intermediário, considerando a base dual, o resultado da função será a reconstrução do estado inicial $|R)$, ou seja, de certa o estado foi teletransportado para Bob.

6.7 Algoritmo da Codificação Superdensa

O algoritmo da codificação superdensa pode ser desenvolvido em Haskell de maneira que não perca a essência semelhante de como foi realizado através do paradigma de programação lógico realizado por James, Ortiz e Sabry (2011) em seu trabalho. A ideia central do algoritmo é a possibilidade de enviar com segurança até quatro informações a partir de um par de mobits emaranhados sem que haja conexão física entre as partes. O algoritmo funciona da seguinte maneira, primeiramente Alice e Bob compartilham um par de mobits emaranhados, neste caso o estado $|R)$, considerando que Alice e Bob estão fisicamente distantes. O primeiro passo é realizar uma transformação no primeiro mobit, para isso utiliza-se uma operação linear garantindo que ela seja especificamente no primeiro mobit.

Define-se as quatro operações de Alice, sendo que dependendo da transformação que realizará em seu mobit afetará diretamente o que Bob medirá como resultado. A forma utilizada para definir essas operações lineares é, primeiro separar o estado $|R)$ e, aplicar a transformação apenas no primeiro mobit, após incorporá-lo ao estado emaranhado novamente com as devidas transformações. Para tal, é necessário fazer uso da expressão lambda, que nesse caso serve para fazer essa ligação novamente ao estado emaranhado.

A função **sd_coding** recebe uma operador linear de dois mobits e um estado emaranhado, tendo como saída uma lista de possíveis mobits resultantes do processo de medição.

```
sd_coding :: Mqbit (Bool,Bool) → Lin(Bool,Bool)(Bool,Bool) →
             [Mqbit(Bool,Bool)]
sd_coding st alice  = measure (st `mbind` alice) basedual
```

Portanto, basicamente dependendo da escolha de Alice (operador linear) será possível transmitir a informação para Bob, ou seja, no momento da medição do processo como um todo, ele terá um resultado diferente do estado inicial.

6.8 Algoritmo Unique-Sat

Nesta seção apresenta-se o desenvolvimento do algoritmo da satisfação booleana Unique-Sat. Este algoritmo consiste em verificar se dada uma função booleana ela é satisfazível ou insatisfazível.

Observa-se que para exemplificar a construção deste algoritmo, estão sendo considerados como entrada três mobits: y, x_1 e x_2, respectivamente y representa o mobit de controle enquanto que x_1 e x_2 são os mobits registro.

Primeiramente define-se o mapeamento *black box* $U_f = |y\rangle\,|\bar{x}\rangle = |y \veebar f(x)\rangle\,|\bar{x}\rangle$, que irá realizar o mapeamento da função $f(x)$ com cada elemento registro da entrada do algoritmo. Note que neste mapeamento, a função f_1 é uma dada função booleana qualquer que deseja-se verificar a satisfabilidade.

```
uf :: Lin (Bool,Bool,Bool) (Bool,Bool,Bool)
uf (y,x1,x2) = orQ (y, (f1 x1))   `mbind` λ a1 →
               orQ (y, (f1 x2))   `mbind` λ a2 →
               orQ (a1, a2)       `mbind` λ a3 →
               mreturn (a3,x1,x2)
```

A operação $\veebar$ é representada através do operador linear orQ que recebe dois valores do

tipo $Bool$ e realiza uma operação relativa a porta clássica *Or-Exclusive* e retorna um mobit.

```
orQ :: Lin (Bool,Bool) Bool
orQ (top,bot) = mreturn ((top || bot) && not (top && bot))
```

Os operadores ss e st são relativos aos operadores s e $s^{\dagger}$ apresentados por Willcock e Sabry (2011). Esses operadores tem como objetivo definir as alterações nos estados dos mobits da seguinte forma:

```
st :: Lin Bool Bool
st False = mobit_0
st True  = mobit_sig

ss :: Lin Bool Bool
ss False = mobit_sig
ss True  = mobit_1
```

Além destes operadores, é necessário definir o operador $cnot$, ou seja, *not* controlado. Basicamente, a alteração no estado ocorrerá se apenas se o mobit controle for $True$.

```
cnot :: Lin (Bool,Bool) Bool
cnot (top,bot) = if top == True
                 then mreturn (not bot)
                 else mreturn bot
```

Após determinados os operadores intermediários, pode-se construir a sequência lógica do algoritmo. Primeiro aplica-se o operador ss nos mobits x_1, x_2. Em seguida realiza-se o mapeamento através do operador U_f. O próximo passo é aplicar o operador st no mobit y e novamente ss nos operadores x_1, x_2. O operador $cnot$ é aplicado no estado inteiro considerando y como mobit controle, por fim, aplica-se o operador st no mobit y e então realiza-se a medida do estado inteiro.

```
unSat :: Lin (Bool,Bool,Bool)(Bool,Bool,Bool)
unSat (y,x1,x2) = ss x1           `mbind` λ x11          →
                  ss x2           `mbind` λ x21          →
                  uf (y,x11,x21)  `mbind` λ (y1,x12,x22)→
                  st y1           `mbind` λ y2           →
                  ss x12          `mbind` λ x13          →
                  ss x22          `mbind` λ x23          →
                  cnot (y2,x13)   `mbind` λ x14          →
                  cnot (y2,x23)   `mbind` λ x24          →
                  st y2           `mbind` λ y3           →
                  mreturn (y3,x14,x24)
```

Após a realização do mapeamento e todas as transformações necessárias, para saber o resultado é fundamental realizar a medida no estado padrão para a base $n + 1$ mobits.

```
measure (y3,x14,x24) (mreturn [(True,True,True)])
```

Logo, se a medida do estado for $|0)\,|\bar{0})$, a função f é dita insatisfazível, por outro lado, se a medida apresentar qualquer outro resultado a função é dita satisfazível.

6.9 Resumo do Capítulo

Neste capítulo, apresenta-se o desenvolvimento da biblioteca MQS, modelo que considera a computação quântica modal desenvolvida na linguagem Haskell. A biblioteca MQS apresenta principalmente abordagens utilizadas no trabalho de Vizzotto, Altenkirch e Sabry (2006), por exemplo, para definir um mobit, utiliza-se a mesma maneira de mapeamento de vetores, contudo no modelo proposto o mapeamento é realizado para campos finitos. Outra característica é utilização de mônadas para realizar as computações com contextos. Os campos finitos, foram definidos a partir de um novo tipo de dados, sendo que as duas operações foram construídas a partir de cálculos modulares. Descreve-se também, toda a estrutura necessária para o desenvolvimento de algoritmos quânticos, finalizando com o desenvolvimento dos algoritmos da Codificação Superdensa, Teleportação Quântica e Unique-Sat.

7 CONCLUSÃO

O modelo de Computação Quântica proposto por Schumacher e Westmoreland proporciona uma simplificação da Computação Quântica Convencional através da substituição de escalares complexos que estão presentes no formalismo de Hilbert por campos finitos. Por consequência, ocorre também a simplificação matemática do modelo quântico, principalmente pela inexistência de ortogonalidade no modelo finito. Contudo observou-se que na prática as características quânticas apresentadas pelo modelo modal estão de acordo com o padrão original.

Evidenciou-se que há limitações de expressão da linguagem Haskell quando existe a necessidade de expressar características puramente quânticas, por exemplo, quando existe a necessidade de trabalhar com um mobit em superposição, ao contrário dos valores *True* e *False*, não existe um tipo nativo na linguagem.

Apesar disso, a simulação da CQ em Haskell apresenta algumas características que possibilitam trabalhar com abstrações e construções de elementos quânticos com facilidade, principalmente, pela forma natural que Haskell lida com a criação de novos tipos de dados, recursividade e mônadas.

Portanto, a construção da biblioteca MQS (*Modal Quantum Simulation*) busca proporcionar um modelo simplificado, trazendo como principal concepção na modelagem do sistema quântico uma estrutura monádica de Computação Quântica Modal em Haskell.

A biblioteca MQS, visa principalmente, proporcionar a novos pesquisadores a possibilidade de explorar o poder de expressão da Teoria Quântica Modal em diferentes áreas de atuação, sendo que a implementação dos algoritmos da Teleportação Quântica, Codificação Superdensa e Unique-Sat servem como exemplos de utilização da biblioteca.

7.1 Trabalhos Relacionados

O trabalho de James, Ortiz e Sabry (2011), denominado *Quantum Computing Over finite fields*, traz uma abordagem da computação quântica utilizando como base a Teoria Quântica Modal de Schumacher e Westmoreland (2011). Neste trabalho investiga-se a proximidade entre programação considerando a TQM e à Lógica Convencional da linguagem Prolog.

Além disso, este trabalho desenvolve uma pequena Linguagem Quântica, com regras de tipos e semântica definida. A base para à estrutura do desenvolvimento da linguagem está no trabalho de Abramsky e Coecke (2004). Outro objetivo que fica claro é a possibilidade de inves-

tigar os fundamentos matemáticos compreendidos no poder Computação Quântica analisando o poder computacional expresso pela Teoria Quântica Modal e comparando com o modelo tradicional cujo formalismo matemático é expresso pelo espaço de Hilbert.

7.2 Trabalhos Futuros

A busca de novos modelos com uma sintaxe mais próxima que é usada por cientistas da computação é uma área que inspira novas pesquisas e trabalhos para seu desenvolvimento.

Não sabe-se como será o comportamento do modelo se necessário implementar algoritmos com uma maior complexidade e envolvendo um maior número de mobits. Portanto, uma sugestão para continuidade de pesquisa é a implementação de novos algoritmos utilizando a biblioteca MQS, além de comparar outros modelos que simulam a computação quântica.

Comparação com outros modelos que simulam a computação quântica.

A paralelização da biblioteca MQS.

Comparação entre o modelo de computação quântica modal e a teoria tradicional da computação quântica.

REFERÊNCIAS

ABRAMSKY, S.; COECKE, B. A Categorical Semantics of Quantum Protocols. In: IN: PROCEEDINGS OF THE 19TH ANNUAL IEEE SYMPOSIUM ON LOGIC IN COMPUTER SCIENCE (LICS'04), IEEE COMPUTER SCIENCE. **Anais...** Press. Arxiv:quant-ph/0402130, 2004.

ALTENKIRCH, T.; GREEN, A. S. **The Quantum IO Monad**. 2009.

BELL, J. S. On the Einstein-Podolsky-Rosen paradox. **Physics**, [S.l.], v.1, p.195–200, 1964.

BENNETT, C. H. et al. **Teleporting an Unknown Quantum State via Dual Classical and EPR Channels**. 1993.

BENNETT, C. H.; WIESNER, S. J. Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. **Phys. Rev. Lett.**, [S.l.], v.69, p.2881–2884, Nov 1992.

DEUTSCH, D. Quantum theory, the Church-Turing principle and the universal quantum computer. **Proceedings of the Royal Society of London**, [S.l.], v.p.97-117, 1985.

EKERT, A.; HAYDEN, P.; INAMORI, H. **Basic Concepts in Quantum Computation**. 2000.

FEYNMAN, R. P. **Feynman Lectures on Computation**. Colorado - The USA: Westview, 2000.

FEYNMAN, R.; SHOR, P. W. Simulating Physics with Computers. **SIAM Journal on Computing**, [S.l.], v.26, p.1484–1509, 1982.

GRUSKA, J. **Quantum computing**. London: McGraw-Hill companies, 1999. (Advanced topics in computer science series).

HANSON, A. J. et al. The Power of Discrete Quantum Theories. **Quantum and Natural Computing Group**, [S.l.], 2011.

HANSON, A. J. et al. Discrete Quantum Theory. , [S.l.], 2013.

HARDY, L. Nonlocality for two particles without inequalities for almost all entangled states. **Physical Review Letters**, [S.l.], v.71, n.11, p.1665–1668, Sept. 1993.

HEUNEN, C.; JACOBS, B. Arrows, like monads, are monoids. In: ANN. CONF. ON MATHEMATICAL FOUNDATIONS OF PROGRAMMING SEMANTICS, MFPS XXII, V. 158 OF ELECTRON. NOTES IN THEORET. COMPUT. SCI, 22. **Proceedings...** Elsevier, 2006. p.219–236.

JAMES, R. P.; ORTIZ, G.; SABRY, A. Quantum Computing over Finite Fields: reversible relational programming with exclusive disjuntions. , [S.l.], 2011.

MOGGI, E. Computational Lambda-calculus and Monads. In: FOURTH ANNUAL SYMPOSIUM ON LOGIC IN COMPUTER SCIENCE, Piscataway, NJ, USA. **Proceedings...** IEEE Press, 1989. p.14–23.

MU, S.-C.; BIRD, R. Functional Quantum Programming. In: IN PROCEEDINGS OF THE 2ND ASIAN WORKSHOP ON PROGRAMMING LANGUAGES AND SYSTEMS. **Anais...** [S.l.: s.n.], 2001.

NIELSEN, M. A.; CHUANG, I. L. **Quantum Computation and Quantum Information**. Cambridge, England: Cambridge Unviersity Press, 2000.

PEYTON-JONES, S. **Haskell 98 language and libraries** : the revised report. Cambridge U.K. ;;New York: Cambridge University Press, 2003.

SABRY, A. Modeling Quantum Computing in Haskell. **Haskell Workshop - Proceedings of the ACM SIGPLAN Workshop on Haskell**, [S.l.], v.p.22, 2003.

SANZONE, B.; SADLER, P. **D-Wave Systems Building Quantum Application Ecosystem, Announces Partnerships with DNA-SEQ Alliance and 1QBit**. 2014.

SCHUMACHER, B.; WESTMORELAND, M. D. Modal Quantum Theory. **Foundations of Physics**, [S.l.], p.918–925, 2011.

SHOR, P. W. Algorithms for quantum computation: discrete logarithms and factoring. In: IN PROC. IEEE SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE. **Anais...** [S.l.: s.n.], 1994. p.124–134.

SOFGE, D. A Survey of Quantum Programming Languages: history, methods, and tools. In: ICQNM. **Anais...** IEEE Computer Society, 2008. p.66–71.

VIZZOTTO, J. K.; ALTENKIRCH, T.; SABRY, A. Structuring quantum effects: superoperators as arrows. **Mathematical Structures in Computer Science**, [S.l.], v.16, n.3, p.453–468, 2006.

WADLER, P. Monads for Functional Programming. In: ADVANCED FUNCTIONAL PROGRAMMING, FIRST INTERNATIONAL SPRING SCHOOL ON ADVANCED FUNCTIONAL PROGRAMMING TECHNIQUES-TUTORIAL TEXT, London, UK, UK. **Anais...** Springer-Verlag, 1995. p.24–52.

WILLCOCK, J.; SABRY, A. Solving UNIQUE-SAT in a Modal Quantum Theory. **School of Informatics and Computing**, [S.l.], 2011.