UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Uma Abordagem Quântica para o Uso de Expressões Regulares

Cheyenne Ribeiro Guedes Isidro

Campina Grande - PB
Fevereiro 2008

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Informática

Uma Abordagem Quântica para o Uso de Expressões Regulares

Cheyenne Ribeiro Guedes Isidro

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Modelos Computacionais e Cognitivos

Bernardo Lula Jr.
(Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

I81a
2008 Isidro, Cheyenne Ribeiro Guedes.
Uma abordagem quântica para o uso de expressões regulares / Cheyenne Ribeiro Guedes Isidro. ─ Campina Grande, 2008.
80f. : il.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Referências.
Orientadores: Prof. Dr. Bernardo Lula Júnior.

1. Teoria da Computação. 2. Computação Quântica. 3. Expressão Regular. 4. Autômato Finito. I. Título.

CDU – 004.423.24(043)

Resumo

As expressões regulares (ER) são conceitos abstratos da Teoria da Computação amplamente utilizados nas tarefas de processamento de texto e casamento de padrão que são aplicadas em diversas áreas, tais como, biologia computacional, processamento de sinais, recuperação de textos, reconhecimento de escrita a mão, reconhecimento de padrões, entre outras. As abordagens clássicas existentes para sua utilização possuem duas fases: i) a transformação da expressão regular em autômato finito, determinístico (AFD) ou não-determinístico (AFN); e ii) a implementação (codificação e simulação) do autômato em hardware clássico. No entanto, essas abordagens são ineficientes na execução de uma de suas fases, seja em relação ao tempo ou ao espaço utilizados. Este trabalho propõe uma abordagem quântica alternativa às abordagens clássicas para o uso de expressões regulares. Na abordagem proposta, a fase de transformação é dividida em duas: a transformação ER-AFN clássica é mantida e introduz-se a transformação do AFN em um autômato finito quântico (AFQ) através da aplicação do algoritmo desenvolvido e apresentado neste trabalho. Esse algoritmo utiliza um modelo de AFQ que reconhece a classe das linguagens regulares (AFQ Ancilla). A transformação é feita em tempo polinomial e preserva o número de estados do AFN, eliminando tanto a ineficiência no uso da memória que resultava da transformação clássica AFN-AFD, quanto a posterior necessidade de minimização. Para a fase de implementação do autômato, é proposto um arcabouço para descrever um autômato finito quântico através da linguagem de circuitos quânticos, utilizando um número de portas polinomialmente proporcional à quantidade de estados do autômato e ao tamanho da palavra de entrada. Assim, a abordagem quântica é computacionalmente eficiente pois ambas as fases possuem complexidade polinomial, tanto de tempo quanto de espaço.

Abstract

Regular expressions (ER) are abstract concepts from Computing Theory widely used in text processing and pattern matching tasks applied in numerous areas, such as, computational biology, signal processing, text retrieving, handwriting recognition, pattern recognition, etc. The classical approaches for their use have two phases: i) transformation from regular expression to finite automata, where the automata can be deterministic (AFD) or non-deterministic (AFN); and ii) automata implementation (codification and simulation) in classical hardware. However, both phases contain inefficiencies related to time or space consumed in their run. This work proposes a quantum approach alternative to the classical ones for regular expression usage. In the proposed approach, the transformation phase is divided into two steps: the classical conversion ER-AFN continues and it is introduced the transformation from AFN to quantum finite automata (AFQ), by using the algorithm developed and presented in this work. This algorithm uses an AFQ model that recognizes the regular language classes, the AFQ Ancilla model. The transformation is carried out in polynomial time and it preserves the AFN's number of states, eliminating both the memory inefficiency resulting from AFN-AFD classical conversion, and the later minimization need. In automata implementation phase, it is proposed a framework to describe a quantum finite automata through the quantum circuits language, using a polynomial number of gates proportional to automata number of states and input word size.

Agradecimentos

À minha família pelo carinho e apoio.

A Nigini Oliveira pela paciência e compreensão.

Ao meu orientador, Bernardo Lula Jr., pela presença constante, reuniões, elogios, puxões de orelha, incentivos e ensinamentos.

Aos amigos sempre presentes nas horas de stress e relax.

A todos que de maneira direta ou indireta contribuíram para a conclusão dessa fase.

À CAPES pelo apoio financeiro na forma de bolsa de mestrado.

Conteúdo

Lista de Símbolos

Q - *conjunto de estados*

Σ - *conjunto de símbolos, alfabeto*

Ω - *alfabeto auxiliar*

δ - *função de transição*

$\hat{\delta}$ - *função de transição estendida*

q_{init} - *estado inicial*

Q_{ac} - *conjunto de estados de aceitação*

w ou u - *uma palavra*

π - *representação em vetor do estado inicial*

η - *representação em vetor do estado final*

X_u - *multiplicação de matrizes correspondentes aos símbolos da palavra* u

M^T - *a matriz transposta de* M

$\mathcal{P}(Q)$ - *conjunto potência do conjunto* Q

$L(A)$ - *linguagem reconhecida pelo autômato* A

$L(E)$ - *linguagem descrita pela expressão regular* E

$L(E_1)L(E_2)$ - *concatenação das linguagens descritas por* E_1 *e* E_2

$L(E_1) \cup L(E_2)$ - *união de linguagens descritas por* E_1 *e* E_2

Σ^* - *operação estrela sobre o alfabeto* Σ

$\bigcup$ - *união de elementos*

$\oslash$ - *conjunto vazio*

ϵ - *símbolo que representa o vazio (nenhum símbolo)*

λ - *probabilidade de ponto de corte para aceitação de palavras*

H - *espaço de Hilbert*

$|\rangle$ - *notação de Dirac para representar um vetor*

$\langle|$ - *notação de Dirac para representar o vetor transposto conjugado*

$|q\rangle$ - *representação quântica do estado* q

$|\Psi\rangle$ - *representação do estado do sistema* Ψ

$|0\rangle_k$ - *registrador com* k *qubits no valor* 0

$|Q|$ - *quantidade de elementos do conjunto* Q

$span\{Q\}$ - *subespaço gerado pelos elementos do conjunto* Q

U - *matriz unitária*

U_a - *matriz unitária correspondente ao símbolo* a

U_w - *multiplicação de matrizes unitárias correspondentes aos símbolos da palavra* w

X, Y, Z, I, $CNOT$, $SWAP$ - *portas quânticas*

$\oplus$ - *operação ou-exclusivo*

$\otimes$ - *produto tensorial*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Este trabalho propõe uma abordagem quântica alternativa às abordagens clássicas para o uso de expressões regulares (ER). As expressões regulares provêm uma maneira fácil e eficiente de expressar palavras de uma linguagem o que permite o seu largo uso como linguagem de entrada para diversas ferramentas e aplicações que realizam pesquisa de texto e casamento de padrões, por exemplo. As abordagens clássicas para o uso das expressões regulares possuem basicamente duas fases: a fase de transformação da expressão regular em um autômato finito (determinístico ou não), ER-AFD ou ER-AFN, e a fase de implementação (codificação ou simulação) do autômato em um hardware clássico. Tem-se, então, duas maneiras de utilização: ER-AFD e implementação de um AFD ou ER-AFN e simulação de um AFN. No entanto, ambas as maneiras possuem deficiências em sua execução, seja em relação ao tempo ou ao espaço consumidos. Na abordagem proposta, a fase de transformação é dividida em duas etapas: transformação ER-AFN e, em seguida, transformação do AFN em um autômato finito quântico (AFQ), AFN-AFQ, segundo um modelo adequado de autômato finito quântico descrito mais além e a partir de um algoritmo eficiente desenvolvido e apresentado neste trabalho. A transformação é feita em tempo polinomial e preserva o número de estados do AFN, eliminando tanto a ineficiência no uso da memória que resultava da transformação clássica AFN-AFD, quanto a posterior necessidade de minimização. Para a fase de implementação, é proposta uma forma sistemática e eficiente de descrever um autômato finito quântico (segundo o modelo mencionado acima) através da linguagem de circuitos quânticos, utilizando um número de portas polinomialmente proporcional à quantidade de estados do autômato e ao tamanho da palavra de entrada. Desse modo, a abordagem quân-

tica proposta é computacionalmente eficiente pois ambas as fases possuem complexidade polinomial, tanto de tempo quanto de espaço

Neste capítulo são apresentadas as informações gerais sobre a pesquisa realizada, definindo o problema abordado, o objetivo do trabalho, a relevância e a metodologia utilizada.

1.1 Autômatos Finitos

A Teoria dos Autômatos (TA) é um ramo da Teoria da Computação que estuda modelos computacionais abstratos através de descrições matemáticas. Seu estudo proporciona o entendimento do que é computação e quais os seus limites, sendo de fundamental importância para os cursos de Ciência da Computação. Mais ainda, proporciona o desenvolvimento de várias habilidades cognitivas no estudante, possibilitando-o expressar, refletir clara e precisamente sobre a resolução de problemas computacionais, assim como perceber as limitações da solução.

Os principais conceitos da Teoria dos Autômatos foram introduzidos na década de 50 como resultado do esforço de diversos pesquisadores incluindo matemáticos, lingüistas, neurofisiologistas e engenheiros eletricistas. Os trabalhos iniciais da área buscavam o desenvolvimento de máquinas que modelassem os processos cognitivos do cérebro humano. O matemático Alan Turing apresentou, em 1936, a Máquina de Turing como sendo um modelo de uma máquina cujo funcionamento era baseado no procedimento usado pelos matemáticos para prova de teoremas [Tur36]. Alguns anos mais tarde, em 1943, McCulloch, psiquiatra, e Pitts, matemático, construíram um modelo para explicar o funcionamento dos neurônios utilizando o conceito de máquinas de estado finito [MP43]. Em 1951, Kleene fez um estudo minucioso sobre o artigo de McCulloch e Pitts, mas somente em 1956, publicou seu trabalho no qual introduziu o conceito de linguagens regulares, originalmente denominadas de “eventos regulares” [Kle56]. Huffman, em 1954, estudando a síntese de circuitos seqüenciais, apresentou a noção de estado de um autômato e de tabela de transição [Huf54]. Moore [Moo56], em 1956, introduziu a noção de estados indistingüíveis e apresentou um algoritmo de minimização, além de desenvolver um modelo de autômato com saída, simultaneamente ao trabalho de Mealy [Mea55]. O conceito de autômatos finitos não-determinísticos foi introduzido por Rabin e Scott [RS59] que expuseram os principais conceitos da área de forma

sistemática, servindo de base para diversos trabalhos posteriores [LP81]. Para uma revisão histórica mais detalhada sugere-se o artigo de Perrin [Per93].

Nas décadas de 60 e 70, os estudos teóricos na área foram bastante ativos. Nas últimas duas décadas, no entanto, o foco das pesquisas tem sido as inúmeras aplicações práticas dos conceitos e idéias da TA que envolvem quase todas as áreas da Ciência da Computação [HMU02], dentre as quais destacam-se:

- softwares verificadores de circuitos digitais [Pet99];
- analisadores léxicos dos compiladores [ASU86];
- diversas aplicações em processamento de linguagem natural, por exemplo, dicionários multilíngüe, *thesauri* e verificadores de escrita, atuando na representação de vocabulários e indexação de textos [RS97, Kar00, KLS98, Arb87, Moh96];
- módulos de casamento de padrões utilizados em diversos softwares de busca [CH97];
- geradores de seqüências de números [Hae03];
- descrições de algoritmos na teoria dos grupos [Sim94b];
- processadores de linguagem XML [Sch07];
- especificações conceituais de *e-Services* [BRSM04];
- verificadores de programas [VW86];
- aplicações em aprendizagem de máquina [Ron95];
- entre outros.

Em virtude desse grande alcance prático, a TA deixou de ser útil apenas em sala de aula, como componente teórico dos cursos de Computação, e tornou-se fundamental para o desenvolvimento de aplicações do mundo real.

No centro da Teoria dos Autômatos está o conceito de Autômato Finito (AF), um conceito matemático abstrato ou um modelo de computação. Mecanicamente, um AF pode ser entendido como uma máquina que possui um controle constituído de estados e um cabeçote de leitura que se move sobre a fita de entrada. A fita é dividida em células em que está escrita

uma seqüência finita de símbolos, chamada de palavra, com um símbolo escrito por célula. No início da computação, o controle do autômato está em um estado especial, dito estado inicial, e o cabeçote posicionado na célula mais à esquerda da fita. Ao ler um símbolo, o autômato altera o estado do controle obedecendo uma função de transição, que depende apenas do estado corrente e do símbolo lido. Após a leitura, o cabeçote move-se para a direita, lê um outro símbolo e altera o estado do controle novamente. Esse processamento é feito sucessivamente até que todos os símbolos escritos na fita sejam lidos. Se ao final da computação, o autômato estiver em um estado especial, dito estado de aceitação, então o autômato *aceita* a palavra lida. Caso contrário, a *rejeita*. O conjunto de todas as palavras que o autômato aceita é dito ser a linguagem que o autômato reconhece. As linguagens reconhecidas por autômatos finitos formam a *classe das linguagens regulares*. Na figura 1.1 encontra-se a representação de um autômato finito evidenciando suas partes constituintes.

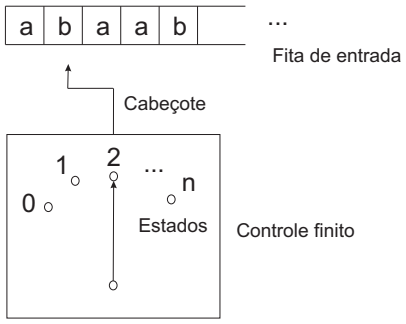


Figura 1.1: Representação do autômato finito

Os AFs atuam deterministicamente quando cada passo da computação segue um único caminho a partir do passo anterior, isto é, para cada combinação de estado e símbolo há um único próximo estado possível. O autômato que possui comportamento determinístico é chamado de Autômato Finito Determinístico (AFD). Em contrapartida, os autômatos podem atuar não-deterministicamente quando a restrição de unitaridade é relaxada, ou seja, quando há mais de um caminho a seguir na escolha do próximo estado. Os Autômatos Finitos Não-determinísticos (AFN) seguem todos os caminhos possíveis simultaneamente. Apesar de parecer mais poderosa no sentido de reconhecer uma classe maior de linguagens, essa extensão não-determinística reconhece a mesma classe de linguagens que os AFDs, sendo, portanto, dois conceitos equivalentes. Além disso, todo AFN pode ser convertido em um AFD equivalente. Geralmente, um AFN é menor que sua contrapartida determinística, tanto

na quantidade de estados quanto na de transições, e seu funcionamento pode ser mais fácil de entender. Além disso, construir um autômato finito não-determinístico é freqüentemente mais fácil que construir um AFD. Essas vantagens justificam o uso dos autômatos finitos não-determinísticos como primeira opção no projeto de soluções utilizando autômatos finitos. Mais ainda, AFNs são bastante utilizados para reduzir a complexidade do trabalho matemático requerido na prova de muitas propriedades importantes da Teoria dos Autômatos.

Os livros sobre Teoria da Computação [CL89, Coh97, HMU02, Koz97, LP81], [Sal69], [Sip97], abordam o tema dos autômatos finitos em maior ou menor profundidade e são uma boa fonte de estudo.

Variações dos modelos básicos AFD e AFN, como por exemplo *tree automata*, *timed automata*, *probabilistic automata,* entre outras, têm sido propostas e são utilizadas em algumas aplicações específicas. *Tree automata*, por exemplo, são utilizados em processadores de linguagem XML [Sch07]; *timed automata* em modelagem de *e-Services* [BRSM04]; e *probabilistic automata* em aprendizagem de máquina [Ron95].

1.2 Expressões Regulares

Um outro conceito importante estudado na Teoria dos Autômatos são as Expressões Regulares (ER), uma notação algébrica para descrever linguagens. Elas são equivalentes aos autômatos finitos no sentido que ambos os conceitos descrevem exatamente a mesma classe de linguagens. Para cada linguagem regular existe, no mínimo, uma ER que a descreve e, no mínimo, um AF que a reconhece.

As expressões regulares oferecem um modo declarativo de expressar as palavras de uma linguagem, provendo uma maneira fácil e eficiente de representar padrões de strings, sejam eles simples ou não. Essa facilidade de representação permite o uso das ERs como linguagem de entrada para as ferramentas de pesquisa de texto, como por exemplo, grep/UNIX, egrep/UNIX, PowerGREP, RegexBuddy [Goy07], além de terem papel fundamental nos componentes léxicos dos compiladores [ASU86] e nas diversas aplicações que realizam casamento de padrões abrangendo as áreas de biologia computacional, processamento de sinais, recuperação de textos, reconhecimento de escrita a mão, reconhecimento de padrões,

entre outras [Nav04].

As ERs têm papel importante também para os programadores. Com seu uso, permite-se uma redução significativa no tempo de codificação já que é possível escrever métodos de busca de padrões em poucas linhas de código. Além disso, uma expressão regular é mais rápida de escrever e mais fácil de depurar e manter que dezenas de linhas de código que realizam a mesma tarefa. Exemplos de linguagens de programação e pacotes que dão suporte ao uso de expressões regulares incluem: Java, JavaScript, .NET, Perl, PHP, Python, Ruby, Tcl, VBScript, VisualBasic 6, Sed e Awk. Perl se destaca entre as demais por ser uma linguagem de programação voltada para o processamento de texto e cujos programas têm as expressões regulares como conceito chave. Para exemplificar o poder das expressões regulares, segue abaixo o programa Perl que pesquisa todos os arquivos .txt do diretório corrente e substitui os valores em Fahrenheit por valores em graus Celsius. O programa consiste em uma linha de código apenas! [Fri02]

```
perl -pi -e 's{([-+]?\d+(\.\d+)?)F\b}{sprintf "$\%.0fC$",$(\dollar 1-32)+5/9}eg' *.txt
```

Os bancos de dados mais utilizados atualmente também possuem funcionalidades baseadas em expressões regulares que podem ser usadas em declarações SQL para filtrar colunas de tabelas. Em alguns é possível também usar expressões regulares para extrair parte de uma coluna ou modificar colunas utilizando a função "localiza-e-substitui". Exemplos de BDs que utilizam ERs incluem MySQL, Oracle e PostgreSQL [Goy07].

Os livros de Teoria da Computação [HMU02, LP81, Sip97] abordam os aspectos teóricos sobre as expressões regulares e a sua relação com autômatos finitos. Para um estudo aprofundado sobre seu uso para programação recomenda-se o livro do Oreilly [Fri02] e o site do Goyvaerts [Goy07].

1.3 Problemas na Utilização de Expressões Regulares

Como apresentado na seção anterior, as expressões regulares são o meio preferido para aplicações que buscam padrões em texto, por exemplo. No entanto, por serem apenas descrições algébricas de linguagens, as ERs não processam palavras sendo necessário, portanto, utilizar um mecanismo que realize tal tarefa: o autômato finito.

Classicamente, existem duas abordagens para utilizar expressões regulares em problemas de busca de padrões e ambas requerem processos de transformação e implementação (codificação e execução). A tabela 1.1 apresenta um resumo dessas abordagens na utilização de expressões regulares que serão detalhadas a seguir e as vantagens e desvantagens de cada uma.

Tabela 1.1: Abordagens na utilização de expressões regulares.

Abordagem	Transformação	Implementação	Vantagem/Desvantagem
1ª	ER - AFN	Simulação	Transformação eficiente, implementação ineficiente
2ª	ER - AFD	Direta	Transformação ineficiente, implementação eficiente

Na primeira abordagem, a transformação ER-AFN é eficiente no sentido de que um dos algoritmos existentes que realiza tal tarefa tem complexidade de tempo da ordem de $O(r)$ e o espaço utilizado durante a aplicação do algoritmo também é da ordem de $O(r)$, onde r é o tamanho da expressão regular a ser transformada [Tho68]. No entanto, a implementação (codificação e execução) do AFN não é realizada de forma direta. É preciso simulá-lo. O método de simulação básico executa em tempo $O(p|Q|^2)$ e espaço $O(|\Sigma||Q|^2)$, onde p é o tamanho da palavra de entrada, $|Q|$ é a quantidade de estados do AFN e $|\Sigma|$ é a quantidade de símbolos do alfabeto. Tal método pode ser considerado uma transformação *on the fly* do AFN para um AFD, onde apenas os estados determinísticos necessários no próximo passo são construídos a partir do estado determinístico atual [Hol00]. Outros métodos de simulação foram desenvolvidos baseados no método básico, como programação dinâmica e bit-paralelismo. Entretanto, eles possuem restrições no uso. A programação dinâmica, por exemplo, não é aplicável a qualquer AFN. Já o bit-paralelismo é utilizado apenas na pesquisa de padrões mais simples e, a cada tipo de padrão pesquisado (classes de caracteres, caracteres opcionais, lacunas) é necessário projetar um algoritmo diferente que simule o NFA [Nav04]. Dependendo do tipo de casamento de padrão a ser realizado (exato ou aproximado) um método pode ser mais eficiente que outro, demandando assim uma análise prévia na escolha do método a ser utilizado.

Já na segunda abordagem, a ineficiência está no processo de transformação ER-AFD pois o mesma não é realizado de forma direta. É preciso transformar a expressão regular em um autômato finito não determinístico (ER-AFN) e posteriormente em um autômato finito determinístico (AFN-AFD) [ASU86]. Enquanto a transformação ER-AFN é eficiente,

como mostrado anteriormente, a transformação AFN-AFD não o é. O principal algoritmo de transformação de AFN para AFD pode levar, no pior caso, a um aumento exponencial no número de estados do autômato [RS59]. Isto é, se o AFN possui n estados, o AFD resultante possuirá 2^n estados. Visando diminuir essa grande quantidade de estados, pode-se converter o AFD em um AFD mínimo. O algoritmo de minimização de melhor desempenho computacional é o algoritmo de Hopcroft apresentado em [Hop71] e [Gri73]. Sua complexidade no tempo é da ordem $O(m\ log\ m)$, onde m é a quantidade de estados do AFD a ser minimizado. Contudo, o crescimento exponencial do número de estados resultante da transformação AFN-AFD torna a minimização um problema *NP-hard* [Ski97], pois o algoritmo de Hopcroft passa a ter complexidade da ordem de $O(2^n\ log\ 2^n)$. Entretanto, uma vez tendo o AFD é possível executá-lo em tempo $O(p)$, onde p é o tamanho da palavra de entrada.

A eficiência desse processo se dá na fase de implementação do AFD pois, por ser determinístico, o autômato é bastante simples de codificar e executar, demandando tempo linear [Hol00] [ASU86].

Mesmo com as deficiências apresentadas, ambas as abordagens têm sido utilizadas nas diversas ferramentas que possibilitam o uso de expressões regulares como linguagem de descrição de padrões de texto. Algumas ferramentas utilizam a primeira abordagem pois nem sempre ocorre a explosão de estados e o AFD tem quase a mesma quantidade de estados do AFN equivalente, não podendo portanto ser aplicada em todos os casos. Em outras ferramentas que utilizam a segunda abordagem, a simulação de AFNs é bastante ineficiente pois utilizam a técnica de *backtracking* (demanda tempo exponencial!), embora como apresentado acima nessa seção, existam métodos de simulação mais eficientes tais como bit paralelismo e programação dinâmica, porém com restrições de uso.

1.4 Abordagem Alternativa Utilizando Hardware Quântico

Este trabalho propõe uma abordagem alternativa para utilização de expressões regulares baseada em um modelo de Autômato Finito Quântico (AFQ) que garante eficiência computacional tanto do processo de transformação quanto do de implementação. Autômatos quânticos são similares aos autômatos clássicos em descrição e funcionamento, mas são intrinseca-

mente não-determinísticos, devido à característica probabilística da Computação Quântica (CQ). Os conceitos básicos da CQ são apresentados no apêndice A.

A Computação Quântica (CQ) é um novo paradigma computacional que faz uso dos efeitos da Mecânica Quântica (MQ), tais como superposição e interferência, para processar informação. Seu estudo iniciou-se na década de 80 quando Feynman [Fey82] afirmou que sistemas clássicos não poderiam simular sistemas quânticos eficientemente, pois tal simulação utilizaria recursos de ordem exponencial, sugerindo a construção de uma máquina que explorasse os princípios da Mecânica Quântica. A partir de então, a CQ desenvolveu-se com contribuições iniciais de Deutsch [Deu85], [Deu89], Bernstein e Vazirani [BV97], Simon [Sim94a], entre outros.

Em 1994, por conseqüência da publicação do algoritmo de Shor [Sho94], que resolve o problema da fatoração de números inteiros grandes em tempo polinomial, as pesquisas na área ganharam impulso e abandonaram o *status* de mera curiosidade acadêmica. Dada a utilidade da fatoração para os sistemas de criptografia atuais, esse algoritmo é considerado o primeiro algoritmo quântico de relevância prática.

Shor utilizou em seu trabalho outro algoritmo quântico importante: a Transformada Quântica de Fourier (QFT - Quantum Fourier Transform). Enquanto sua versão clássica mais eficiente, a Transformada Rápida de Fourier (FFT - Fast Fourier Transform), possui complexidade na ordem de O($N\ log\ N$), a QFT possui complexidade na ordem de O($log^2 N$), sendo exponencialmente mais rápida que aquela [Mar06].

Outro algoritmo de destaque na área é o algoritmo de Grover [Gro96]. Desenvolvido para realizar buscas em uma lista (base de dados) desordenada, seu ganho em complexidade é quadrático em relação aos algoritmos clássicos existentes. Apesar do ganho não ser tão expressivo quanto o conseguido pela QFT, este algoritmo de busca é utilizado, sobretudo, para acelerar a solução de problemas cuja única estratégia clássica é o método da força bruta, a exemplo dos problemas NP-completos [KFPL07] e de problemas da Teoria da Informação [ONA07, OA07].

As pesquisas utilizando o paradigma da Computação Quântica têm avançado no desenvolvimento de algoritmos que sejam mais eficientes que os clássicos, como os apresentados anteriormente, provendo ótimos resultados e incentivando novos estudos na área. Entretanto, a busca por algoritmos quânticos mais eficientes pode não ter resultados expressivos

em determinadas áreas da Computação. Por exemplo, os problemas abrangidos pela teoria dos autômatos finitos (linguagens regulares) são resolvidos em tempo linear no contexto clássico [LP81], tornando desnecessário o uso de autômatos finitos quânticos para a solução desses problemas. Todavia, é possível utilizar os conceitos e elementos da Computação Quântica para melhorar outros aspectos da Computação Clássica, como por exemplo, o uso de expressões regulares, que é o problema abordado nessa dissertação.

A abordagem alternativa quântica proposta neste trabalho, e que tem como requisitos a eficiência computacional tanto do processo de transformação quanto do de implementação, consiste das seguintes etapas:

1. Transformação clássica ER-AFN;

2. Transformação do AFN-AFQ, segundo o modelo de autômato finito quântico utilizado, apresentado no capítulo 4, utilizando o algoritmo desenvolvido nesse trabalho. Este algoritmo executa em tempo polinomial preservando o número de estados do autômato;

3. Codificação do AFQ, resultante da segunda etapa, na linguagem dos circuitos quânticos e execução do modelo em um "hardware" quântico. Essa codificação usa um número de portas quânticas proporcional ao tamanho da palavra e à quantidade de estados do autômato, e a execução ocorre em tempo proporcional ao tamanho da palavra e independente do número de estados do autômato.

Como o processo de transformação AFN-AFQ preserva o número de estados do autômato, o problema de explosão de estados que resulta da transformação AFN-AFD clássica é eliminado, bem como a necessidade de minimização do autômato. A implementação (codificação e execução) também é eficiente pois do ponto de vista da codificação o tamanho do circuito é linear no tamanho da entrada e no número de estados demandando, e, do ponto de vista da execução, o algoritmo executa em tempo linear ao tamanho da entrada e independente da quantidade de estados do autômato.

A tabela 1.2 apresenta um resumo das abordagens clássicas e da abordagem quântica proposta.

Tabela 1.2: Abordagens clássicas e abordagem quântica proposta.

Abordagem	Transformação	Implementação	Vantagem/Desvantagem
1ª	ER - AFN	Simulação	Transformação eficiente, implementação ineficiente
2ª	ER - AFD	Direta	Transformação ineficiente, implementação eficiente
Proposta	ER - AFN - AFQ	Direta (via circuito quântico)	Transformação eficiente, implementação eficiente

1.5 Objetivos do Trabalho

O objetivo deste trabalho é a concepção de uma abordagem alternativa eficiente para o uso de expressões regulares. A solução proposta passa pela definição de um algoritmo de transformação de um AFN em um AFQ em tempo polinomial preservando o número de estados do AFN original, eliminando a ineficiência no uso da memória que resultava da transformação clássica AFN-AFD e posterior minimização. Em adição, é proposto um arcabouço ou forma sistemática de descrever um autômato finito quântico através da linguagem de circuitos quânticos utilizando um número de portas polinomialmente proporcional à quantidade de estados do autômato e ao tamanho da palavra de entrada.

1.6 Relevância

A utilização da Computação Quântica para solução de problemas do mundo clássico mostra-se bastante promissora, uma vez que os efeitos quânticos permitem, em determinados casos, um ganho computacional expressivo. Contudo, a CQ pode atuar também diminuindo a complexidade do projeto de autômatos ou máquinas. O algoritmo desenvolvido neste trabalho é um resultado que atesta tal ganho, ao reduzir significativamente a complexidade do processo de construção da solução (autômato). A larga utilização prática dos autômatos finitos justifica a importância das pesquisas na área em busca de métodos, técnicas e algoritmos que facilitem seu uso.

Além disso, o arcabouço proposto para implementar um autômato finito quântico utilizando a linguagem de circuitos quânticos é bastante genérico. Para implementar um AFQ qualquer segundo o modelo de autômato utilizado basta realizar duas alterações no circuito. A primeira é construir as portas que implementam as matrizes correspondentes aos símbolos do alfabeto sendo utilizado, para que elas correspondam à função de transição do autômato, e substituí-las nos locais onde aparecem no circuito. O algoritmo descrito por Nielsen e

Chuang [NC00] pode ser utilizado para construir tais portas. A segunda alteração é aumentar ou diminuir o número de qubits para representar os estados do autômato e os símbolos do alfabeto. Desse modo, qualquer que seja o AFQ a ser implementado, o arcabouço proposto sofrerá pouquíssimas mudanças, tornando a implementação de autômatos quânticos escalável.

Outro ponto importante é que máquinas quânticas mais simples, como os AFQs, devem estar disponíveis bem antes que os almejados computadores quânticos, pois elas demandam apenas a implementação física de poucas portas (de um e dois qubits), já descritas e estudadas pela CQ [BBC$^+$95]. Ainda, é possível que as primeiras implementações de computadores quânticos não sejam completamente quânticas e uma parte clássica opere juntamente com uma parte quântica.

É necessário destacar que este trabalho de Mestrado está inserido no âmbito das pesquisas do IQuanta - Instituto de Estudos em Informação e Computação Quânticas - e contribuirá para a consolidação do mesmo como centro de pesquisa na área, ao apresentar mais um resultado positivo no uso da CQ.

1.7 Metodologia

Por ser um trabalho teórico, a metodologia adotada foi composta de atividades de estudo e pesquisa apresentadas a seguir:

1. Realização da revisão bibliográfica sobre os grupos de pesquisa cujo enfoque são os autômatos finitos quânticos.

2. Estudo dos autômatos finitos quânticos existentes, dando ênfase à descrição, classe de linguagem reconhecida, pontos positivos e negativos de cada modelo.

3. Estudo da teoria dos autômatos finitos clássicos buscando problemas ou ineficiências de uma abordagem clássica.

4. Elaboração e formalização de uma alternativa para o problema descrito na etapa anterior utilizando uma abordagem quântica.

5. Elaboração de artigos técnico-científicos para divulgação do trabalho.

Todas essas atividades foram documentadas e seus artefatos estão disponíveis no *site* http://www.dsc.ufcg.edu.br/∼cha/qfa, incluindo os relatórios das atividades 2 e 3, a proposta de dissertação resultante da atividade 4 e o artigo técnico-científico intitulado "Um algoritmo para transformar um autômato finito não-determinístico em um autômato finito quântico preservando o número de estados e a linguagem reconhecida", publicado nos Anais do Workshop-Escola de Computação e Informação Quântica - WECIQ2006 contendo os resultados iniciais da pesquisa [IL06].

1.8 Estrutura da Dissertação

Este capítulo apresentou as informações gerais sobre a pesquisa, definindo o problema abordado, os objetivos do trabalho, a relevância e a metodologia utilizada.

No Capítulo 2 são apresentados os Autômatos Finitos (AF) clássicos abordando a definição formal dos mesmos, sua utilização e os modelos determinístico (AFD) e não-determinístico (AFN), assim como o algoritmo de transformação AFN-AFD. Em seguida, as Expressões Regulares (ER) e sua relação com os autômatos finitos são apresentadas. Os algoritmos de transformação ER-AFN e ER-AFD são discutidos ao fim do capítulo.

No Capítulo 3 são abordados o conceito e os modelos de Autômatos Finitos Quânticos, discorrendo sobre suas características e a busca por um modelo que reconheça a classe das linguagens regulares.

A descrição da abordagem quântica proposta para o uso de expressões regulares é apresentada no início do Capítulo 4 e suas etapas são detalhadas em seguida. São apresentados o algoritmo de transformação de AFN para AFQ e a forma sistemática de implementar um autômato finito quântico utilizando a linguagem de circuitos quânticos. No fim do capítulo, é apresentada a análise de complexidade da abordagem.

As conclusões e sugestões para trabalhos futuros são apresentadas no Capítulo 5.

Por fim, o Apêndice A possui uma pequena introdução à Computação Quântica apresentando os conceitos básicos utilizados na alternativa proposta e sua leitura é de fundamental importância para os leitores que não estiverem familiarizados com notações e conceitos da Mecânica Quântica. O Apêndice B descreve sucintamente o conceito de família de circuitos e os atributos de avaliação de eficiência dos mesmos. O Apêndice C apresenta em detalhes

uma definição utilizada no capítulo 4.

Capítulo 2

Autômatos Finitos e Expressões Regulares

Para o entendimento da abordagem proposta é necessário detalhar alguns conceitos básicos apresentados no capítulo introdutório. Neste capítulo são apresentados os Autômatos Finitos (AF) clássicos abordando a definição formal dos mesmos, sua utilização e os modelos determinístico (AFD) e não-determinístico (AFN), assim como o algoritmo de transformação AFN-AFD. Em seguida, as Expressões Regulares (ER) e sua relação com os autômatos finitos são apresentadas. Os algoritmos de transformação ER-AFN e ER-AFD são discutidos ao fim do capítulo.

2.1 Autômatos Finitos Determinísticos (AFD)

Formalmente [HMU02], um autômato finito determinístico (AFD) é uma 5-tupla $A = (Q, \Sigma, \delta, q_{init}, Q_{ac})$ em que:

- Q é um conjunto finito de estados;
- Σ é um conjunto finito de símbolos de entrada, chamado de alfabeto;
- $\delta : Q \times \Sigma \rightarrow Q$ é a função de transição, que define as regras de mudança entre os estados da máquina;
- $q_{init} \in Q$ é o estado inicial;

- $Q_{ac} \subseteq Q$ é um conjunto de estados de aceitação, que pode ser unitário ou não.

O autômato A inicia sua computação no estado inicial q_{init} e lê uma palavra $w = w_1 w_2 \dots w_n$ da esquerda para a direita, símbolo por símbolo. No i-ésimo passo, estando em um estado q, A lê o símbolo w_i e atualiza seu estado para $p = \delta(q, w_i)$. A máquina é dita determinística pois cada passo da computação leva a um único estado seguinte.

A **aceita** a palavra w se depois da leitura de todos os símbolos de w, o estado no qual o autômato pára pertence a Q_{ac}. Diz-se que um autômato A **reconhece** a linguagem formada por todas as palavras aceitas. A classe de linguagens reconhecida pelos autômatos finitos determinísticos é a classe das linguagens regulares.

Representação

Pode-se representar os autômatos através de diagramas de estados, tabelas de estados e através de matrizes.

Os **diagramas de estados** são a representação gráfica dos autômatos mais utilizada. Um autômato $A = (Q, \Sigma, \delta, q_{init}, Q_{ac})$ consiste em um grafo direcionado em que:

- Cada estado $q \in Q$ possui um nó correspondente;
- Para cada $q \in Q$ e para cada símbolo $a \in \Sigma$, seja $\delta(q, a) = p$. Então existe um arco direcionado do nó q para o nó p, rotulado por a;
- O estado inicial q_{init} é indicado por uma seta, e
- Os estados de aceitação são marcados por círculos duplos.

Para o autômato $A = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_2\})$ cujo δ é:

$$\begin{aligned} \delta(q_0, a) &= q_2 \quad \delta(q_2, a) = q_4 \\ \delta(q_0, b) &= q_1 \quad \delta(q_2, b) = q_0 \\ \delta(q_1, a) &= q_3 \quad \delta(q_3, a) = q_3 \\ \delta(q_1, b) &= q_1 \quad \delta(q_3, b) = q_0 \end{aligned}$$

tem-se o diagrama apresentado na figura 2.1.

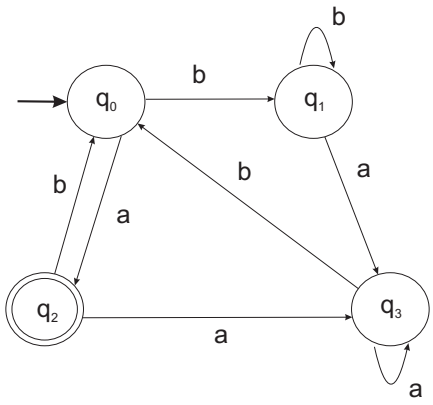


Figura 2.1: Representação do AFD A por diagrama de estado

As **tabelas de transição** de estados são outra forma de representação que consiste em uma tabela descrevendo a função de transição δ do autômato. As linhas da tabela correspondem aos estados e as colunas correspondem aos símbolos de entrada. A entrada na tabela para a linha correspondente ao estado q, e para a coluna correspondente ao símbolo a é o estado $\delta(q, a)$.

Para o autômato $A = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_2\})$ cujo δ é:

$$\delta(q_0, a) = q_2 \quad \delta(q_2, a) = q_4$$
$$\delta(q_0, b) = q_1 \quad \delta(q_2, b) = q_0$$
$$\delta(q_1, a) = q_3 \quad \delta(q_3, a) = q_3$$
$$\delta(q_1, b) = q_1 \quad \delta(q_3, b) = q_0$$

tem-se a tabela de transição 2.1 correspondente.

Tabela 2.1: Representação do AFD A por tabela de transição de estados

	a	b
q_0	q_2	q_1
q_1	q_3	q_1
q_2	q_3	q_0
q_3	q_3	q_0

Para ilustrar a computação de uma palavra w utilizando essa representação, seja o autômato A definido anteriormente e $w = aba$. Tem-se então os seguintes passos da computação.

Passo 1: $\delta(q_0, a) = q_2$

Passo 2: $\delta(q_2, b) = q_0$

Passo 3: $\delta(q_0, a) = q_2$

Como o autômato A termina a computação no estado de aceitação q_2 então a palavra w é aceita por A.

A **notação matricial** permite uma manipulação fácil durante a computação da palavra. Um autômato $A = (Q, \Sigma, \delta, q_{init}, Q_{ac})$ é descrito como segue:

- O estado inicial é representado através de vetor-coluna onde cada linha corresponde a um estado $q \in Q$ e sua entrada será 1 se corresponder ao estado inicial, e 0 caso contrário.
- Os estados finais são representados por um único vetor-coluna onde cada linha corresponde a um estado $q \in Q$, e sua entrada será 1 se corresponder a um estado final, e 0 caso contrário
- Para cada símbolo a do alfabeto existe uma matriz de transição X_a onde as linhas e colunas correspondem aos estados do autômato. A entrada para a linha correspondente ao estado q, e para a coluna correspondente ao estado q' é 1 se $\delta(q, a) = q'$, e 0 caso contrário.

Para calcular se uma dada palavra w é aceita ou não pelo autômato, utiliza-se a expressão 2.1:

$$ACC_u = \pi^T X_u \eta = \begin{cases} 1 & \text{se a palavra é aceita pelo autômato} \\ 0 & \text{caso contrário} \end{cases} \tag{2.1}$$

em que:

- π é a representação em vetor do estado inicial;
- η é a representação em vetor do estado final; e
- X_u é a multiplicação das matrizes correspondentes aos símbolos da palavra u.

Para o autômato $A = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_2\})$ cujo δ é:

$$\begin{aligned} \delta(q_0, a) &= q_2 \quad \delta(q_2, a) = q_4 \\ \delta(q_0, b) &= q_1 \quad \delta(q_2, b) = q_0 \\ \delta(q_1, a) &= q_3 \quad \delta(q_3, a) = q_3 \\ \delta(q_1, b) &= q_1 \quad \delta(q_3, b) = q_0 \end{aligned}$$

tem-se a seguinte representação:

$$\pi = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \eta = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad X_a = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad X_b = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Para a palavra $w = ab$, tem-se o seguinte cálculo:

$$\begin{aligned} ACC_u &= \pi^T X_u \eta \\ &= \pi^T X_a X_b \eta \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \\ &= 0 \end{aligned}$$

Como $ACC = 0$, a palavra $w = ab$ não é aceita pelo autômato.

2.2 Autômatos Finitos Não-Determinísticos (AFN)

Numa máquina determinística, ao ler um símbolo da entrada, sabe-se exatamente qual será o próximo estado alcançado, como mostrado nos AFDs na seção anterior 2.1. Numa máquina não-determinística, a leitura de um símbolo pode levar a vários estados diferentes. Um autômato finito não-determinístico (AFN), difere do AFD apenas pela função de transição que possui, pois esta permite ir a zero, um ou mais estados diferentes ao ler o mesmo símbolo.

Formalmente, [HMU02] descreve um AFN como uma 5-tupla $A = (Q, \Sigma, \delta, q_{init}, Q_{ac})$, onde:

- Q é um conjunto finito de estados;
- Σ é um conjunto finito de símbolos de entrada, chamado de alfabeto;
- $\delta : Q \times \Sigma \rightarrow 2^Q$ é a função de transição, que define as regras de mudança entre os estados da máquina;
- $q_{init} \in Q$ é o estado inicial;
- $Q_{ac} \subseteq Q$ é o conjunto de estados de aceitação, que pode ser unitário ou não.

A função de transição δ de A retorna um conjunto de estados pertencentes a Q, diferentemente do AFD, cujo retorno era um único estado apenas.

Ao ler uma palavra u, o AFN A tem, portanto, mais de um caminho a seguir e ele segue todos os caminhos simultaneamente. A aceitação da palavra acontece se, ao final da leitura, algum estado final é alcançado por qualquer um dos caminhos percorridos. O conjunto de todas as palavras aceitas pelo AFN A constitui a linguagem aceita pelo autômato.

Seja $A = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$ um AFN cuja função δ é representada pela tabela de transição 2.2. A representação por diagrama de estados é mostrada na figura 2.2:

Tabela 2.2: Representação do AFN A por tabela de transição de estados

	a	b
q_0	$\{q_0, q_1\}$	q_0
q_1	$\oslash$	q_2
q_2	$\oslash$	$\oslash$

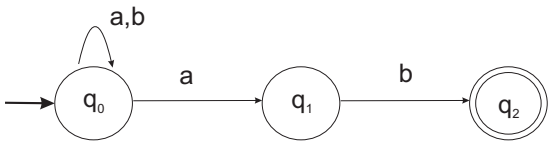


Figura 2.2: Representação do AFN A por diagrama de estado

Para entender melhor o processo de aceitação, segue abaixo o passo-a-passo do processamento da palavra $u = aabab$ para o AFN A definido anteriormente.

Passo 1:	$\delta(q_0, a)$	$= \{q_0, q_1\}$	
Passo 2:	$\delta(q_0, a) \cup \delta(q_1, a)$	$= \{q_0, q_1\} \cup \oslash$	$= \{q_0, q_1\}$
Passo 3:	$\delta(q_0, b) \cup \delta(q_1, b)$	$= \{q_0\} \cup \{q_2\}$	$= \{q_0, q_2\}$
Passo 4:	$\delta(q_0, a) \cup \delta(q_2, a)$	$= \{q_0, q_1\} \cup \oslash$	$= \{q_0, q_1\}$
Passo 5:	$\delta(q_0, b) \cup \delta(q_1, b)$	$= \{q_0\} \cup \{q_2\}$	$= \{q_0, q_2\}$

Ao fim do processamento, verifica-se que o autômato encontra-se nos estados $\{q_0, q_2\}$. Logo, como um dos estados é de aceitação (q_2), a palavra $u = aabab$ é aceita.

Como visto na seção anterior 2.1, os AFDs são restritos quanto à aceitação de linguagens reconhecendo apenas a classe das linguagens regulares (as mais simples). Ao relaxar a restrição imposta pela função de transição de determinar apenas um único estado por vez, como foi feito no AFN, esperava-se que a classe de linguagens deste modelo fosse mais abrangente. No entanto ambos são modelos equivalentes e reconhecem a mesma classe de linguagens [HMU02] [LP81].

Apesar de não serem mais abrangentes que os AFDs, os AFNs são úteis como ferramenta inicial na resolução de problemas envolvendo autômatos pois, na maioria das vezes, sua construção é mais rápida, seu funcionamento é mais simples de entender e são mais compactos (menos estados) que seus equivalentes determinísticos. Devido a essas características, os AFNs são bastante utilizados também no trato formal da teoria facilitando a prova de teoremas [HMU02].

Para ilustrar a conveniência na utilização de um AFN, considere a linguagem L descrita pela expressão regular $(ab \cup aba)^*$. O menor autômato finito determinístico [LP81] que a reconhece é apresentado na figura 2.3 a seguir:

Em contrapartida, a mesma linguagem L é reconhecida pelo autômato finito não-

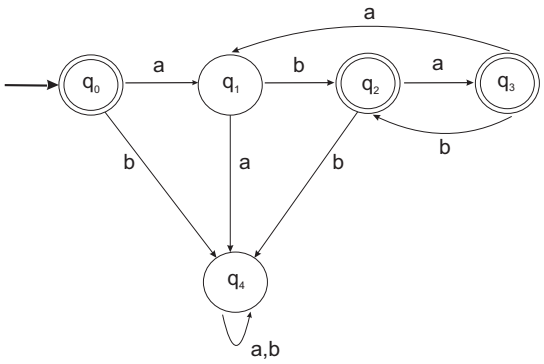


Figura 2.3: Menor AFD que reconhece L

determinístico apresentado na figura 2.4:

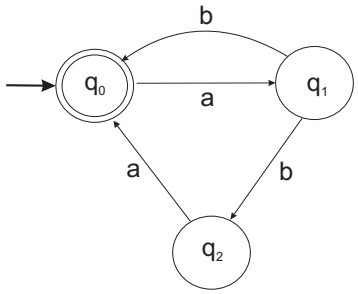


Figura 2.4: AFN que reconhece L

Percebe-se que o AFN é bem mais simples que o AFD equivalente, pois possui menos estados (3 contra 5 do AFD) e transições (4 contra 10 do AFD), levando ao entendimento mais rápido do seu funcionamento e, conseqüentemente, à verificação mais rápida que a linguagem que ele reconhece é a linguagem L descrita pela expressão regular $(ab \cup aba)^*$.

2.2.1 Autômatos Finitos Não-Determinísticos com Transições ε (ε-AFN)

Existe uma variante do modelo AFN que permite transições ε, isto é, transições sobre o string vazio. Tal modelo é chamado de ε-AFN pode executar a transição sem receber nenhum símbolo de entrada. Essa nova característica também não aumenta a classe de linguagens reconhecida pelos autômatos, mas dá mais flexibilidade na construção dos mesmos, principalmente quando relacionados às expressões regulares, como será visto na seção 2.4.

2.3 Transformação de AFN em AFD

Para cada autômato finito não-determinístico existe, no mínimo, um autômato finito determinístico que reconhece a mesma linguagem. A idéia da transformação entre autômatos é visualizar o AFN como ocupando a cada momento, não um estado, mas um conjunto de estados. A transformação de um AFN em um AFD equivalente é apresentada a seguir, segundo descrito em [HMU02] e [Sip97].

Seja $A = (Q, \Sigma, \delta, q_{init}, Q_{ac})$ um AFN reconhecendo uma linguagem L. Deseja-se construir um AFD $B = (Q', \Sigma, \delta', \{q'_{init}\}, Q'_{ac})$ tal que $L(A) = L(B)$. O alfabeto Σ é o mesmo para os dois autômatos. B é construído como segue:

1. $Q' = \mathcal{P}(Q)$

   O conjunto de estados de B, Q' é o conjunto potência de Q, isto é, o conjunto de todos os subconjuntos de Q. Se Q possui n estados, Q' possuirá 2^n estados.

2. Para $S \in Q'$ e $a \in \Sigma$ seja $\delta'(S, a) = \{q \in Q | q \in \delta(s, a)$ para algum $s \in S\}$.

   Para calcular $\delta'(S, a)$ é necessário examinar todos os estados $s \in S$ e verificar para quais estados $q \in Q$ o autômato A vai ao estar em s e ler o símbolo a. De posse desses estados, faz-se a união deles e atribui ao valor do $\delta'(S, a)$. Outra forma de escrever essa expressão é $\delta'(S, a) = \bigcup_{s \in S} \delta(s, a)$.

3. $q'_{init} = \{q_{init}\}$.

   O estado inicial de B corresponde ao conjunto $S \in Q'$ que contém apenas o estado inicial de A.

4. $Q'_{ac} = \{S \in Q' | S$ contém um estado de aceitação de $A\}$. O conjunto de estados de aceitação de B é formado pelos subconjuntos $S \in Q'$ que contém no mínimo um estado de aceitação de A, isto é, $S \cap Q_{ac} \neq \oslash$.

Para facilitar o entendimento, segue um exemplo. Seja o autômato $A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$ um AFN cuja função δ é representada pela tabela de transição 2.3. A representação por diagrama de estados é mostrada na figura 2.5.

O AFD equivalente B será:

1. $Q' = \{\oslash, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$

Tabela 2.3: Tabela de transição de estados do AFN A de exemplo

	0	1
q_0	$\{q_0, q_1\}$	q_0
q_1	$\oslash$	q_2
q_2	$\oslash$	$\oslash$

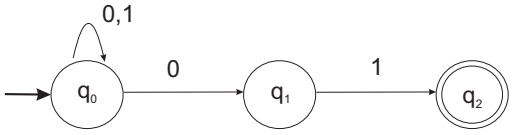


Figura 2.5: Representação do AFD A de exemplo

2. Função de transição apresentada na tabela 2.4.

Tabela 2.4: Representação do AFD B por tabela de transição de estados

	0	1
$\oslash$	$\oslash$	$\oslash$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\oslash$	$\{q_2\}$
$\{q_2\}$	$\oslash$	$\oslash$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1, q_2\}$	$\oslash$	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

3. $q'_{init} = \{q_0\}$

4. $Q'_{ac} = \{\{q_2\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$

Percebe-se que dos oito estados apresentados na tabela 2.4, começando no estado inicial de B, só é possível alcançar os estados $\{q_0\}, \{q_0, q_1\}, \{q_0, q_2\}$, ou seja, os outros cinco estados são inacessíveis e podem ser descartados, diminuindo a quantidade de estados do autômato. É possível construir a tabela já eliminando esses estados inacessíveis, seguindo o algoritmo de *construção de subconjuntos* [Nik07]. Tal algoritmo pode ser utilizado também

em ε-AFNs pois ele trata dos estados alcançados através de transições ε. Para o entendimento do algoritmo, as operações utilizadas são descritas a seguir:

- ε-fechamento(q) = conjunto dos estados do AFN alcançáveis a partir do estado $q \in Q$ apenas a partir de transições-ε.
- ε-fechamento(Q) = conjunto dos estados do AFN alcançáveis a partir de algum estado $q \in Q$ apenas a partir de transições-ε
- $move(Q, a)$ = conjunto dos estados do AFN para os quais existe uma transição a partir de algum estado $q \in Q$ lendo o símbolo de entrada a.

O algoritmo de construção de subconjuntos é apresentado a seguir.

Construção de subconjuntos

Seja ε-fechamento(q_{init}) o único estado no conjunto Q' do AFD A
while existir estados não marcados $S \in Q'$ **do**
  marque S
  for cada símbolo $a \in \Sigma$ **do**
    $U = \varepsilon$-fechamento$(move(S, a))$
    if U não está em Q' **then**
      adicione U em Q' como estado não marcado
    end if
    $\delta'(S, a) = U$
  end for
end while

Utilizando o algoritmo apresentado acima, a tabela de transição será criada contendo apenas os estados alcançáveis a partir do estado inicial do autômato.

Para o autômato A utilizado como exemplo, o AFD equivalente B possui três estados apenas e está representado no diagrama de estados da figura 2.6.

Percebe-se que com essa construção, pode haver uma redução no número de estados do autômato resultante se existirem subconjuntos de estados que não são alcançáveis (o que é comum), no entanto, no pior caso, é possível que todos os estados sejam alcançáveis e o AFD resultante possua 2^n estados. Desse modo, a transformação AFN-AFD é ineficiente

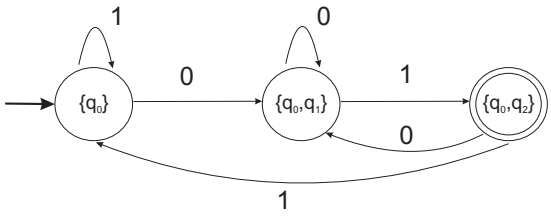


Figura 2.6: Representação do AFD B equivalente

pois a complexidade de espaço será da ordem $O(2^n)$ e, conseqüentemente, o tempo gasto para construir o autômato determinístico também será exponencial.

2.4 Expressões Regulares (ER)

As Expressões Regulares (ER) são representações algébricas de linguagens, e assim como os autômatos finitos, descrevem a classe das linguagens regulares. A linguagem representada por uma expressão regular E é denotada por $L(E)$.

Formalmente [Sip97], E é uma expressão regular se E é:

1. ε;
2. $\oslash$;
3. a, para qualquer símbolo a do alfabeto Σ.
4. $(E_1 + E_2)$, onde E_1 e E_2 são expressões regulares;
5. $(E_1 \cdot E_2)$, onde E_1 e E_2 são expressões regulares ou
6. $(E_1{}^*)$, onde E_1 é uma expressão regular.

Os itens 1 e 3 definem as expressões regulares que representam as linguagens $L(\varepsilon) = \{\varepsilon\}$ e $L(a) = \{a\}$, respectivamente. O item 2 define a expressão regular que representa a linguagem vazia, $L(\oslash) = \oslash$. Já os itens 4, 5 e 6 definem as expressões formadas a partir de outras expressões e cujas linguagens são obtidas a partir da união $L(E_1 + E_2) = L(E_1) \cup L(E_2)$ ou concatenação $L(E_1 \cdot E_2) = L(E_1) \cdot L(E_2)$ de duas outras linguagens, ou a partir da operação estrela $L(E_1^*) = (L(E_1))^*$ de uma linguagem, respectivamente. [1]

[1] As operações de união, concatenação e estrela são operações sobre linguagens. A classe das linguagens regulares é fechada sob essas operações.

Por exemplo, a expressão regular 0^*10^* representa a linguagem cujas palavras contém um único 1, enquanto a expressão regular $(01^*) + (10^*)$ representa a linguagem cujas palavras possuem um único 0 seguido por qualquer quantidade de 1s ou possuem um único 1 seguido por qualquer quantidade de 0s.

Os parênteses podem ser suprimidos das expressões, embora sejam bastante úteis para redefinir a precedência das operações, caso não se deseje seguir as regras existentes, que para as expressões regulares, o operador estrela é o de mais alta precedência, seguido pelo operador de concatenação e, por fim, o de união.

2.5 Algoritmos de Transformação de ER para AF

As expressões regulares são equivalentes aos autômatos finitos pois ambos os formalismos definem a mesma classe de linguagens. Ainda mais, é possível realizar as transformações entre as representações, ER-AF e AF-ER. No entanto, dependendo do tipo do autômato (AFD ou AFN) o processo de transformação será mais, ou menos, complicado. No problema abordado nesse trabalho, o interesse são as transformações ER-AF já que investiga-se a utilização das expressões regulares. Nas subseções a seguir são apresentados e analisados os algoritmos de transformação entre ER-AFN e ER-AFD.

2.5.1 Algoritmos de Transformação de ER para AFN

O algoritmo clássico mais conhecido para realizar a transformação ER-AFN foi desenvolvido por Thompson, em 1968, e diversas variantes foram desenvolvidas com o passar do tempo. A versão apresentada a seguir é bastante simples e pode ser encontrada em alguns livros que abordam o tema das expressões regulares, tais como [HMU02], [Sip97] e [ASU86]. A entrada para o algoritmo é uma expressão regular E sobre um alfabeto Σ e a saída será um ε-AFN A aceitando $L(E)$.

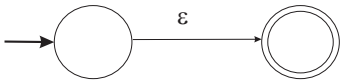
Construção de Thompson

Primeiro divide-se E em suas sub-expressões constituintes. Em seguida, usando as regras 1 e 2 abaixo, constrói-se um AFN para cada um dos símbolos básicos em E (aqueles que são ε ou um símbolo do alfabeto). Os símbolos básicos correspondem aos itens 1 e 2 da definição de expressões regulares descritos na seção anterior. É importante destacar que se

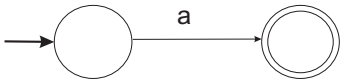
um símbolo a ocorre várias vezes em E, um AFN separado é construído para cada ocorrência do símbolo. Para a expressão $\oslash$, o autômato é criado usando a regra 3.

Então, guiado pela estrutura sintática da expressão regular E, combina-se esses AFNs indutivamente usando a regra 4 abaixo até que se obtenha um AFN para a expressão inteira. Cada AFN intermediário produzido durantes o curso da construção corresponde à uma sub-expressão de E e tem várias propriedades importantes: ela possui exatamente um estado final, nenhuma aresta chega ao estado inicial e nenhuma aresta sai do estado final.

1. Para ε, construa o seguinte AFN



2. Para $a \in \Sigma$, construa o seguinte AFN



3. Para $\oslash$, construa o seguinte AFN

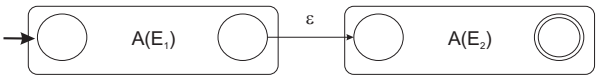
4. Suponha que $A(E_1)$ e $A(E_2)$ são AFNs para as expressões regulares E_1 e E_2.

a) Para a expressão regular $(E_1 + E_2)$ construa o seguinte AFN composto $A(E_1 + E_2)$

Existe uma transição-ε do novo estado inicial para o estados iniciais de $A(E_1)$ e $A(E_2)$. Existe uma transição-ε a partir dos dos estados finais de $A(E_1)$ e $A(E_2)$ para o novo estado final. Os estados iniciais e finais de $A(E_1)$ e $A(E_2)$ não são iniciais ou finais para o autômato composto. Note que qualquer caminho do estado inicial para o estado final deve passar através de $A(E_1)$ ou $A(E_2)$ exclusivamente. Assim, o AFN construído reconhece

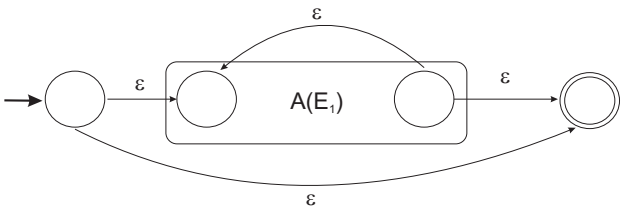
$L(E_1) \cup L(E_2)$.

b) Para a expressão regular $(E_1 \cdot E_2)$, construa o seguinte AFN composto $A(E_1 \cdot E_2)$:



O estado inicial de $A(E_1)$ se torna o estado inicial do AFN composto, e o estado final de $A(E_2)$ se torna o estado final do AFN composto. Entre o estado final de $A(E_1)$ e o estado inicial de $A(E_2)$ é adicionada uma transição-ε e os estados perdem o status de final e inicial no AFN composto. Desse modo, um caminho do estado inicial para o estado final do autômato composto deve necessariamente passar pelo autômato $A(E_1)$ e em seguida pelo autômato $A(E_2)$, de modo que o rótulo do caminho seja dado por uma palavra em $L(E_1) \cdot L(E_2)$. Assim, o autômato composto construído reconhece $L(E_1) \cdot L(E_2)$.

c) Para a expressão regular $({E_1}^*)$, construa o seguinte AFN composto $A({E_1}^*)$:



Aqui o autômato permite ir diretamente do estado inicial para o final através de uma transição-ε, representando o fato que $\varepsilon \in L({E_1}^*)$, independente da expressão E_1. Permite também ir do estado inicial ao final passando por $A(E_1)$ uma ou mais vezes, representando as palavras das linguagens $L(E_1)$, $L(E_1)L(E_1)$, $L(E_1)L(E_1)L(E_1)$, e assim por diante. Desse modo, o autômato $A({E_1}^*)$ construído reconhece $L({E_1}^*)$

d) Para a expressão regular (E_1), utilize o próprio $A(E_1)$ como AFN pois os parênteses na expressão não alteram sua linguagem.

Sempre que um novo estado é construído, ele é rotulado com um nome distinto. Dessa forma, dois estados de qualquer AFN componente não pode ter o mesmo nome. Mesmo se o mesmo símbolo aparecer diversas vezes em E_1, deve ser criado para cada instância do símbolo um AFN separado com seus próprios estados.

Esta construção é feita em tempo $O(n)$, onde n é o tamanho da expressão regular a ser convertida e o espaço utilizado também tem complexidade linear. O autômato resultante é

um ε-AFN com m estados, onde m é no máximo $2n$ já que a cada passo do algoritmo são adicionados no máximo dois novos estados ao autômato.

Para converter um ε-AFN em um AFN comum (sem transições-ε) gasta-se tempo $O(m^3)$ [HMU02]. O algoritmo utilizado é similar ao algoritmo de construção de subconjuntos exposto na seção 2.3 mas a quantidade de estados não é alterada. Segundo [HSW01], é possível ainda converter uma expressão regular diretamente em um AFN equivalente, sem transições-ε, em tempo $O(n^2 log n)$. Sendo assim, a transformação ER-AFN é computacionalmente eficiente dado que existem algoritmos com complexidade de tempo de ordem polinomial e de espaço de ordem linear ao tamanho da entrada.

2.5.2 Algoritmos de Transformação de ER para AFD

Enquanto as transformações ER-AFN e ER-ε-AFN são computacionalmente eficientes, como apresentadas anteriormente, a transformação ER-AFD não o é. O procedimento existente para realizar esse tipo de conversão consiste em duas etapas: converter a expressão regular em um autômato finito não-determinístico (com ou sem transições-ε) e em seguida converter o autômato finito não-determinístico em um autômato finito determinístico.

A primeira etapa foi discutida anteriormente e mostra-se eficiente em tempo e espaço. No entanto a transformação AFN-AFD da segunda etapa é ineficiente, como apresentado na seção 2.3. No pior caso, o AFD resultante terá um número exponencial de estados e a complexidade de tempo e espaço será exponencial.

Capítulo 3

Autômatos Finitos Quânticos

Neste capítulo são abordados o conceito e os modelos de Autômatos Finitos Quânticos, discorrendo sobre suas características Para aqueles não familiarizados com os conceitos básicos da Computação Quântica, recomenda-se a leitura prévia do apêndice A.

3.1 Visão Geral dos Autômatos Finitos Quânticos

A contrapartida quântica para os modelos de máquina apresentados no capítulo 2 são os Autômatos Finitos Quânticos (AFQ). A diferença principal em relação aos Autômatos Finitos Clássicos é que em um AFQ os estados de Q podem estar em superposição. Usando a base computacional e uma enumeração dos estados em Q, pode-se identificar o estado q_i com o i-ésimo vetor da base $|q_i\rangle$ (usando a notação de Dirac) e representar uma superposição de estados como um vetor $|q\rangle = \sum_{i=0}^{|Q|} \alpha_i |q_i\rangle$, onde $q_i \in Q$, α_i é a amplitude (um número complexo) do estado q_i e $\sum_{i=0}^{|Q|} |\alpha_i|^2 = 1$ é a condição de normalização.

De um modo geral, um AFQ é um autômato com memória finita cujos estados podem estar em superposição e a computação ocorre através da aplicação de operadores unitários, correspondendo à função de transição dos autômatos clássicos. Para conhecer o resultado da computação é preciso realizar medições no estado do autômato. Um AFQ pode ser visto como um modelo híbrido entre o autômato finito não-determinístico, devido à sua característica de superposição, e o autômato probabilístico, pois a cada estado existe associada uma probabilidade de leitura.

Os modelos de AFQ propostos, tais como [MC97], [KW97], [Pas00], [NIHK02], diferem

basicamente pela forma de leitura da fita de entrada, e pela forma de medição. Quando se permite movimento do cabeçote apenas para a direita ao ler a palavra, tem-se o modelo chamado de *1-way*, e quando se permite movimento tanto para a direita como esquerda, tem-se o modelo *2-way*. Se a medição do estado do autômato ocorre apenas ao final da leitura da fita, tem-se o modelo *Measure once* (MO), mas se são feitas medições projetivas depois de cada símbolo lido, tem-se o modelo *Measure many* (MM). Essas variações na definição do autômato acarretam em variações na classe de linguagens que o modelo reconhece.

O reconhecimento de uma linguagem por AFQs pode ocorrer de diversas formas descritas a seguir. As variações ocorrem devido à ausência ou presença de um valor de ponto de corte (limite) e de uma margem de erro associada.

- Uma linguagem L é dita ser aceita por um AFQ A com probabilidade $p > 1/2$, se cada palavra $x \in L (x \notin L)$ é aceita (rejeitada) com probabilidade no mínimo p. Nesse caso, não há margem de erro.

- Uma linguagem L é dita ser aceita por um AFQ A com ponto de corte $\lambda > 1/2$, se para todo $x \in L (x \notin L)$ a probabilidade de A aceitar (rejeitar) x é $p > \lambda (\leq \lambda)$. Esse caso é similar ao anterior, pois o valor de λ é $1/2$, embora seja permitido atribuir outros valores para λ. Não é estabelecida nenhuma margem de erro.

- Uma linguagem L é dita ser aceita por um AFQ A com ponto de corte $\lambda > 1/2$ e erro limitado, se existe um $\epsilon > 0$ tal que qualquer $x \in L (x \notin L)$ é aceita (rejeitada) por A com probabilidade $\geq \lambda + \epsilon$ ($\leq \lambda - \epsilon$). Se A aceita uma linguagem L com ponto de corte λ, mas não com erro limitado, então A é dita aceitar L com erro ilimitado. Nessa forma de aceitação, o erro é levado em consideração.

- Uma linguagem L é dita ser aceita com erro ilimitado por um lado por um AFQ A, se A aceita todas as palavra em L com certeza ($p = 1$), e rejeita palavras não em L com uma probabilidade $p > 0$ (ou vice versa).

Os modelos pioneiros são os de 1-way de uma única medição (1-way MO-AFQ) desenvolvido por Moore e Crutchfield [MC97], e os modelos de 1-way e 2-way de múltiplas medições (1-way MM-AFQ e 2-way MM-AFQ) desenvolvidos por Kondacs e Watrous [KW97]

e são apresentados nas seções 3.2 e 3.3. Outros modelos serão comentados na seção 3.3.1 e o modelo utilizado na solução do problema é abordado na seção 3.4.

3.2 1-way MO-AFQ

O primeiro modelo de autômato quântico foi apresentado por [MC97] e é chamado de *Measure-once quantum finite automata* (1-way MO-AFQ). Este modelo permite a movimentação do cabeçote de leitura da fita apenas para a direita (1-way) e a medição do estado é feita ao fim da leitura, sendo um extensão natural do modelo clássico.

Formalmente, um 1-way MO-AFQ é uma 5-tupla $A = (H, \Sigma, U_\sigma, |q_{init}\rangle, H_{ac})$ em que:

- H é um espaço de Hilbert n-dimensional, onde os estados são vetores nesse espaço;
- Σ é um alfabeto de entrada;
- U_σ são matrizes unitárias de transições de dimensão n para cada símbolo $\sigma \in \Sigma$, que representam a função de transição do autômato;
- $|q_{init}\rangle$ é um vetor estado inicial de dimensão n onde $|q_{init}|^2 = 1$;
- $H_{ac} \subseteq H$ é um subspaço de aceitação com um operador de projeção P_{ac}.

A probabilidade do autômato A aceitar a palavra w é dada por:

$$P(w) = |{q_{init}}^T \cdot U_w \cdot P_{ac}|^2 \tag{3.1}$$

onde $U_w = U_{w_1} \cdot U_{w_2} \ldots U_{w_n}$

O autômato inicia no estado $|q_{init}\rangle$, e à medida que cada símbolo w_i da palavra é lido, aplica-se a matriz unitária U_{w_i} correspondente. Ao final da leitura da palavra é feita uma medição projetiva no estado do autômato, utilizando o operador P_{ac}, e a norma é calculada. Essa computação resultará na probabilidade $P(w)$ do estado resultante estar no subespaço de aceitação. Se $P(w) > 0$ então A aceita a palavra w, caso contrário A rejeita a palavra.

Para ilustrar o funcionamento do autômato, segue um exemplo. Seja o 1-way MO-AFQ A definido por: $H = |q_0\rangle, |q_1\rangle$, $\Sigma = \{0, 1\}$, $q_{init} = |q_0\rangle$, $H_{ac} = P_{ac} = |q_1\rangle \langle q_1|$. A função

de transição é representada pelas matrizes U_0 e U_1, cuja atuação nos estados do autômato é apresentada a seguir:

$$\begin{aligned} U_0 |q_0\rangle &= \frac{1}{\sqrt{2}}|q_0\rangle + |q_1\rangle \\ U_0 |q_1\rangle &= |q_1\rangle \\ U_1 |q_0\rangle &= |q_0\rangle \\ U_1 |q_1\rangle &= |q_0\rangle \end{aligned}$$

Uma outra forma de apresentar a função de transição é através do δ, similar aos autômatos finitos clássicos. Cada entrada da função, $\delta(q_i, a, q_j)$, significa a probabilidade do autômato estar no estado q_i, ler o símbolo a e passar ao estado q_j. Para o exemplo, tem-se a seguinte função de transição.

$$\begin{aligned} \delta(q_0, 0, q_0) = \frac{1}{\sqrt{2}} \qquad \delta(q_0, 1, q_0) = 1 \\ \delta(q_0, 0, q_1) = \frac{1}{\sqrt{2}} \qquad \delta(q_0, 1, q_1) = 0 \\ \delta(q_1, 0, q_0) = 0 \qquad \delta(q_1, 1, q_0) = 1 \\ \delta(q_1, 0, q_1) = 1 \qquad \delta(q_1, 1, q_1) = 0 \end{aligned}$$

Para processar a palavra $w = 01$, o autômato A age como descrito a seguir

1. A inicia no estado $|q_0\rangle$ e aplica a matriz U_0, que corresponde ao primeiro símbolo da palavra w, resultando na superposição de estados $\frac{1}{\sqrt{2}}|q_0\rangle + |q_1\rangle$.

2. Em seguida, aplica a matriz U_1 correspondendo ao segundo símbolo da palavra w. O estado resultante é $\frac{1}{\sqrt{2}}|q_0\rangle + |q_0\rangle = |q_0\rangle$

3. Depois de processada a palavra, é realizada a medição do estado utilizando o projetor $|q_1\rangle \langle q_1|$. Como o estado $|q_0\rangle$ não é de aceitação, a probabilidade resultante é $P(w) = 0$

Assim, a palavra $w = 01$ não é aceita pelo autômato A, pois a probabilidade $P(w)$ não é maior que 0 (zero).

Devido à restrição de unitaridade dos operadores U, este modelo é menos poderoso que os autômatos finitos clássicos, reconhecendo apenas uma subclasse das linguagens regulares. Por exemplo, a linguagem regular $L_{bb} = \{u \in \{a, b\}^* | u$ não contém a substring $bb\}$ não

é reconhecida por um MO-AFQ [MC97]. Brodsky e Pippenger [BP99] mostraram que a classe de linguagens reconhecidas por *1-way MO-AFQ* com erro limitado é exatamente a classe reconhecida pelos autômatos finitos de grupo, chamada linguagens de grupo. Com erro ilimitado, o *1-way MO-AFQ* pode reconhecer linguagens não regulares como $L = \{x \in \{a, b\}^* | |x|_a = |x|_b\}$, que é a linguagem formada por todas as palavras que possuem a mesma quantidade de símbolos a e b, tais como ab, $abab$, $abba$, etc.

3.3 1-way MM-AFQ

O modelo de autômato que permite múltiplas medições, chamado de *Measure-many quantum finite automata* (MM-AFQ) foi introduzido por [KW97] com duas variantes: a leitura da fita pode ser feita apenas para a direita (1-way MM-AFQ), como no modelo da subseção anterior, ou o cabeçote pode mover-se tanto para a direita quanto para a esquerda (2-way MM-AFQ).

O modelo 1-way MM-AFQ é definido como uma 6-tupla $A = (Q, \Sigma, \delta, q_{init}, Q_{ac}, Q_{rej})$ em que:

- Q é um conjunto finito de estados;
- Σ é um alfabeto de entrada e $\Gamma = \Sigma \cup \{\varsigma, \$\}$ é o alfabeto da fita, em que ς e $\$(\notin \Sigma)$ são os símbolos de marcação para o início e fim da fita respectivamente;
- $\delta : Q \times \Gamma \times Q \times D \rightarrow C$ é uma função de transição, em que $D = \{1\}$. D indica o movimento do cabeçote, que nesse modelo é sempre para a direita, representado por 1;
- $q_{init} \in Q$ é o estado inicial;
- $Q_{ac} \subseteq Q$ é um conjunto de estados de aceitação;
- $Q_{rej} \subseteq Q$ é um conjunto de estados de rejeição.

Os elementos de Q_{ac} e Q_{rej} são estados de parada e os elementos de $Q_{non} = Q - (Q_{ac} \cup Q_{rej})$ são estados de não-parada. Assume-se que $q_{init} \in Q_{non}$ e $Q_{ac} \cap Q_{rej} = \oslash$.

Para cada $q, q' \in Q$, $\sigma \in \Gamma$ e $d \in D$, $\delta(q, \sigma, q', d)$ representa a amplitude com a qual a máquina estando no estado q e lendo o símbolo σ, vai ao estado q' movendo o cabeçote para a direita.

Associa-se a cada símbolo $\sigma \in \Gamma$, a matriz V_σ, que deve ser unitária e satisfazer:

$$\delta(q, \sigma, q', d) = \begin{cases} \langle q'|V_\sigma|q\rangle & d = 1 \\ 0 & d \neq 1 \end{cases}$$

Leva-se em consideração os valores de $d \neq 1$ pois é necessário construir toda as linhas da matriz V_σ, mesmo que algumas delas não reflitam transições válidas no autômato.

A restrição de unitaridade de V_σ para cada símbolo σ pode ser verificada utilizando:

$$\sum_{q'} \overline{\langle q'| V_\sigma |q_1\rangle} \langle q'| |V_\sigma |q_2\rangle = \begin{cases} 1 & q_1 = q_2 \\ 0 & q_1 \neq q_2 \end{cases}$$

O autômato A inicia no estado $|q_{init}\rangle$, e à medida que cada símbolo w_i da palavra é lido, aplica-se a matriz unitária V_{w_i} correspondente e move-se o cabeçote na direção d, que é sempre 1. Após a aplicação da matriz observa-se o novo estado através de um projetor nos subespaços ortogonais formados por $E_{ac} = \{|q\rangle : q \in Q_{ac}\}$, $E_{rej}\{|q\rangle : q \in Q_{rej}\}$ e $E_{non} = \{|q\rangle : q \in Q - (Q_{ac} \cup Q_{rej})\}$. Se o estado obtido após a medição pertencer a E_{ac} a palavra é aceita, se pertencer a E_{rej} a palavra é rejeitada, mas se o estado pertencer a E_{non} a computação continua e o próximo símbolo é lido.

Para facilitar o entendimento, seja o 1-way MM-AFQ $A = (Q, \Sigma, \delta, q_{init}, Q_{ac}, Q_{rej})$, onde $Q = \{q_0, q_1, q_{ac}, q_{ref}\}$, $\Sigma = \{a\}$, $q_{init} = q_0$, $Q_{ac} = \{q_{ac}\}$ e $Q_{rej} = \{q_{rej}\}$. A função de transição pode ser apresentada pela ação das matrizes V_a como segue:

$$\begin{aligned} V_a(|q_0\rangle) &= \frac{1}{2}|q_0\rangle + \frac{1}{2}|q_1\rangle + \frac{1}{\sqrt{2}}|q_{rej}\rangle \\ V_a(|q_1\rangle) &= \frac{1}{2}|q_0\rangle + \frac{1}{2}|q_1\rangle - \frac{1}{\sqrt{2}}|q_{rej}\rangle \\ V_\$(|q_0\rangle) &= |q_{rej}\rangle \\ V_\$(|q_1\rangle) &= |q_{ac}\rangle \end{aligned}$$

Algumas transições não foram especificadas, como por exemplo, $V_a(q_{ac})$, porque tais valores não são importantes para definir o funcionamento do autômato. No entanto, eles podem ser definidos arbitrariamente tornando a matriz V_a unitária [AF98].

Pode-se também representar a função de transição pelo δ, mas o processo é mais longo. Por exemplo, $V_a(|q_0\rangle) = \frac{1}{2}|q_0\rangle + \frac{1}{2}|q_1\rangle + \frac{1}{\sqrt{2}}|q_{rej}\rangle$ seria

$$\delta(q_0, a, q_0) = \frac{1}{2} \qquad \delta(q_0, a, q_1) = \frac{1}{2}$$
$$\delta(q_0, a, q_{ac}) = 0 \qquad \delta(q_0, a, q_{rej}) = \frac{1}{\sqrt{2}}$$

Para a palavra $w = aa$ o autômato executa como descrito a seguir.

1. O autômato inicia no estado $|q_0\rangle$ e V_a é aplicada, resultando em $\frac{1}{2}|q_0\rangle + \frac{1}{2}|q_1\rangle + \frac{1}{\sqrt{2}}|q_{rej}\rangle$. Em seguida, a medição é executada. Com probabilidade $1/\sqrt{2}^2 = 1/2$, o estado de rejeição $|q_{rej}\rangle$ é observado e a computação termina rejeitando a palavra. Caso contrário, com probabilidade $1/2$, um estado de não-parada, $|q_0\rangle$ ou $|q_1\rangle$, é observado, e a superposição colapsa para o subespaço $\frac{1}{2}|q_0\rangle + \frac{1}{2}|q_1\rangle$. A computação continua.

2. Agora na superposição de estados $\frac{1}{2}|q_0\rangle + \frac{1}{2}|q_1\rangle$, a matriz V_a é aplicada novamente resultando em

$$\begin{aligned} V_a(\frac{1}{2}|q_0\rangle + \frac{1}{2}|q_1\rangle) &= \frac{1}{2}V_a|q_0\rangle + \frac{1}{2}V_a|q_1\rangle \\ &= (\frac{1}{2}|q_0\rangle + \frac{1}{2}|q_1\rangle + \frac{1}{\sqrt{2}}|q_{rej}\rangle) + (\frac{1}{2}|q_0\rangle + \frac{1}{2}|q_1\rangle - \frac{1}{\sqrt{2}}|q_{rej}\rangle) \\ &= \frac{1}{2}|q_0\rangle + \frac{1}{2}|q_1\rangle \end{aligned}$$

   Após a aplicação de V_a, a medição é executada e um estado de não-parada é observado com probabilidade 1, pois não há nenhum estado de aceitação ou rejeição na superposição.

3. A palavra então termina de ser lida e aplica-se a matriz $V_\$$, resultando em

$$\begin{aligned} V_\$(\frac{1}{2}|q_0\rangle + \frac{1}{2}|q_1\rangle) &= \frac{1}{2}V_\$|q_0\rangle + \frac{1}{2}V_\$|q_1\rangle \\ &= \frac{1}{2}|q_{rej}\rangle + \frac{1}{2}|q_{ac}\rangle \end{aligned}$$

   Com probabilidade $1/2^2 = 1/4$, o estado de rejeição é observado. Com probabilidade também de $1/4$, o estado de aceitação é observado.

A probabilidade total de aceitação para a palavra $w = aa$ é $1/4$, e a probabilidade total de rejeição é $1/2 = 1/4 = 3/4$. Se a condição de aceitação for $P(w) > 0$, então $w = aa$

é aceita pelo autômato A. No entanto, se for estabelecida a aceitação com ponto de corte $\lambda = 1/2$, então $w = aa$ é rejeitada, pois a probabilidade total de aceitação ($1/4$) é inferior ao limite estabelecido.

O modelo *1-way MM-AFQ* tem sido largamente estudado. Kondacs e Watrous [KW97] mostraram que com erro limitado, tal modelo reconhece uma subclasse das linguagens regulares. A linguagem regular $L = \{a, b\}^* a$, por exemplo, não é reconhecida por esse modelo. Posteriormente, Brodsky e Pippenger [BP99] mostraram propriedades de fechamento da classe de linguagens reconhecida com erro limitado enquanto Valdats [Val00] provou que a classe reconhecida com ponto de corte por *1-way MM-AFQ* não é fechada por união, isto é a união de duas linguagens reconhecidas com determinado ponto de corte não resulta em uma linguagem reconhecida com o mesmo ponto de corte. Ambainis et al. [AKV01] desenvolveram uma série de condições suficientes (e necessárias) para que certas linguagens regulares não sejam aceitas por tal modelo, generalizando os critérios definidos anteriormente por Brodsky e Pippenger [BP99] e Ambainis e Freivalds [AF98]. Essas condições dizem respeito à existência de 'construções proibidas' nos DFAs mínimos que reconhecem uma dada linguagem regular.

Ambainis e Freivalds [AF98] mostraram que quanto menor a probabilidade de aceitação requerida, maior a classe de linguagens reconhecida, ao provar que existem linguagens que podem ser reconhecidas com probabilidade 0.68, mas não com probabilidade $7/9$. Ambainis et al. [ABFK99] estenderam esse resultado construindo uma hierarquia de linguagens na qual cada linguagem pode ser reconhecida com probabilidade menor que a anterior. Ambainis e Kikusts [AK01] foram além ao determinar as probabilidades máximas de aceitação por *1-way MM-AFQ* para diversas linguagens.

Em relação ao espaço de estados dos autômatos, Ambainis e Freivalds [AF98] e Bonner et al. [BFR02] provaram que para algumas linguagens o *1-way MM-AFQ* pode ser exponencialmente menor que sua contrapartida clássica. Nessa mesma linha, Kikuts [Kik98] constrói um *1-way MM-AFQ* que é quadraticamente menor que qualquer DFA equivalente para uma dada linguagem. Por outro lado, Midrijanis [Mid02] apresenta uma linguagem em que o DFA que a reconhece utiliza $O(n)$ estados, enquanto o AFQ precisa de no mínimo de $2^{\Omega(n^{\log n})}$ estados. Ou seja, em alguns casos, o autômato quântico pode ser exponencialmente maior que o determinístico equivalente.

3.3.1 Demais Modelos

Os modelos pioneiros apresentados anteriormente mostraram-se inadequados para o uso na abordagem quântica proposta, pois, para efetuar a transformação AFN-AFQ da segunda etapa, era necessário que o modelo de autômato finito quântico utilizado reconhecesse no mínimo a classe das linguagens regulares. Outros modelos foram pesquisados analisando a classe de linguagens que reconhecem.

Ambainis et al. [ABF$^+$99] introduziram a noção de autômato finito quântico multifita (QFMA) como sendo uma generalização do modelo *1-way MM-AFQ* com múltiplas fitas, e provaram que existe uma linguagem reconhecida por tal modelo que não é reconhecida por autômato finito determinístico ou probabilístico. Posteriormente, foi introduzido por Ambainis e Watrous [AW02], o modelo de autômato finito de *2-way* com estados clássicos e quânticos (2QDFA). Esta é uma variação do modelo *2-way MM-AFQ* cujos estados podem incluir estados quânticos, mas cuja movimentação de leitura é requerida ser clássica. Em relação à capacidade computacional, são apresentadas duas linguagens para as quais o 2QCFA é mais eficiente que o autômato clássico de *2-way*. Amano e Iwana [AI99] apresentaram o modelo *1.5-way MM-AFQ* onde a movimentação de leitura da fita para a esquerda não é permitida. A classe de linguagens para tal modelo não é bem definida embora seu poder seja semelhante aos dos autômatos de pilha. Bertoni et al. [BMP03] introduziram o modelo de autômato finito quântico de *1-way* com linguagem controle (1QFC). Assim como o *1-way MM-AFQ*, o 1QFC permite observações após cada símbolo lido mas a aceitação da palavra só ocorre se o resultado da computação pertencer a uma dada linguagem controle (que é uma linguagem regular). As linguagens aceitas com ponto de corte isolado por 1QFC são regulares.

Estes novos modelos encontrados eram variantes dos modelos pioneiros mas também não satisfaziam à condição necessária de reconhecimento das linguagens regulares. Por fim, a busca resultou em dois modelos similares que ambos reconhecem no mínimo as linguagens regulares. Ciamarra [Cia01] apresenta um modelo quântico reversível de autômato finito, e mostra que sua capacidade computacional é no mínimo igual aos dos autômato finitos determinísticos clássicos, reconhecendo toda a classe das linguagens regulares. Nessa mesma direção, Paschen [Pas00] apresenta um modelo de autômato finito quântico usando qubits auxiliares (AFQ Ancilla) e que reconhece todas as linguagens regulares. Ambos os mode-

los poderiam ter sido utilizados na abordagem proposta neste trabalho pois ambos atendem ao requisito necessário, no entanto, devido à simplicidade e à similaridade com o modelo pioneiro 1-way MO-AFQ, o modelo AFQ Ancilla foi o escolhido.

3.4 AFQ Ancilla

A restrição de unitaridade das transições é um fator limitante para os autômatos finitos quânticos pois reduz a classe de linguagens que reconhecem. Visando contornar tal deficiência, Paschen [Pas00] introduziu um alfabeto auxiliar e alterou a função de transição do modelo 1-way MO-AFQ de Moore e Crutchfield [MC97], definindo um novo modelo denominado AFQ Ancilla.

Formalmente, um AFQ Ancilla A é uma 6-tupla $A = (Q, \Sigma, \Omega, \delta, q_{init}, Q_{ac})$ em que:

- Q é um conjunto finito de estados;
- Σ é um alfabeto de entrada;
- Ω é um alfabeto auxiliar;
- $\delta : Q \times \Sigma \times Q \times \Omega \rightarrow C[0,1]$ é a função de transição;
- $q_{init} \in Q$ é o estado inicial;
- $Q_{ac} \subseteq Q$ é um conjunto de estados de aceitação;

A função de transição δ define matrizes unitárias de acordo com a restrição:

$$\sum_{q \in Q, w \in \Omega} \overline{\delta(q_1, \sigma, q, w)} \delta(q_2, \sigma, q, w) = \begin{cases} 1 & q_1 = q_2 \\ 0 & q_1 \neq q_2 \end{cases}$$

$\forall \sigma \in \Sigma$, e $q_1, q_2 \in Q$.

O funcionamento do autômato é idêntico ao do modelo 1-way MO-AFQ permitindo uma única medição ao final da leitura da palavra. Este modelo de autômato finito quântico é no mínimo tão "poderoso" quanto os modelos AFD e AFN clássicos pois Paschen em [Pas00] mostra que dado um AFD A mínimo que reconhece uma linguagem regular L, é possível

construir um AFQ Ancilla B que reconhece L com probabilidade 1. Devido a essa característica, o modelo AFQ Ancilla é utilizado na abordagem quântica proposta nesse trabalho para transformar um autômato finito clássico em um autômato finito quântico.

Capítulo 4

Abordagem Quântica Para o Uso de Expressões Regulares

Neste capítulo é apresentada a abordagem quântica proposta para o uso de expressões regulares. Na seção 1 é apresentada uma visão geral da abordagem proposta e suas etapas são detalhadas nas seções 2 e 3. Na seção 4 é feita a análise da abordagem avaliando a complexidade do algoritmo de transformação de autômatos utilizado na segunda etapa, e a complexidade da construção e execução do circuito utilizado na terceira etapa.

4.1 Visão Geral da Abordagem Quântica Proposta

Como discutido no capítulo 1, seção 1.3, as abordagens clássicas para a utilização de expressões regulares apresentam ineficiências tanto na fase de transformação quanto na fase de implementação do autômato. A abordagem alternativa quântica visa eficiência computacional em todo o processo e consiste das seguintes etapas:

1. Transformação clássica da expressão regular em um autômato finito não-determinístico;
2. Tranformação do AFN, resultante da primeira etapa, em um autômato finito quântico, segundo o modelo AFQ Ancilla, apresentado no capítulo 3, seção 3.4;
3. Implementação do AFQ, resultante da segunda etapa, na linguagem dos circuitos quânticos e execução do modelo em um “hardware” quântico.

A primeira etapa da abordagem consiste na utilização do algoritmo clássico apresentado no capítulo 2, seção 2.5.1, de modo que não será detalhada nesse capítulo. As segunda e terceira etapas são apresentadas em detalhes nas seções a seguir.

4.2 Segunda Etapa: Transformação AFN-AFQ Ancilla

A partir do AFN resultante da primeira etapa da abordagem, aplica-se um algoritmo para transformá-lo em um autômato finito quântico (AFQ) do tipo AFQ Ancilla (vide capítulo 4, seção 3.4). A idéia do algoritmo é definir cada um dos elementos do AFQ Ancilla a partir dos elementos do AFN. Os detalhes dessa transformação são explicados a seguir.

Algoritmo: Dado um AFN qualquer $A = (Q, \Sigma, \delta, q_{init}, Q_{ac})$, definir um Ancilla AFQ $B = (Q', \Sigma, \Omega, \delta', q'_{init}, Q'_{ac})$ seguindo os passos abaixo:

1. Para cada um dos estados clássicos deve ser definido um estado quântico correspondente, representado por vetores na notação de Dirac. Caso o autômato não-determinístico possua alguma transição que leve ao conjunto vazio ($\oslash$), é necessário adicionar um novo estado ao conjunto de estados do AFQ.

   Formalmente, $Q' = \{|q_i\rangle : q_i \in Q, 0 \leq i < n\} \cup \{|q_n\rangle\}$, onde o estado $|q_n\rangle$ é adicionado apenas se o δ de A possuir alguma transição não definida ($\oslash$).

2. O alfabeto Σ não sofre alteração, no entanto, o alfabeto auxiliar do AFQ não tem correspondente clássico e é definido como o alfabeto binário $\Omega = \{0, 1\}$

3. Após definir os estados e os alfabetos, é preciso definir a função de transição no novo autômato. No AFQ Ancilla, o δ' corresponde à função $\delta' : Q \times \Sigma \times Q \times \Omega \rightarrow C[0, 1]$, que associa uma probabilidade de mudança ao estado. O não-determinismo clássico foi representado no AFQ por uma distribuição uniforme de probabilidades, e cada uma das entradas da função deve ser definida segundo a equação a seguir.

   Para cada entrada da função $\delta(q_i, a) = \{q_{j_i}, \cdots, q_{j_k}\}$ definir:

$$\begin{aligned} \delta'(q_i, a, q_{j_d}, 0) &= \frac{1}{\sqrt{|\delta(q_i,a)|}} && \text{, para todo } d, 1 \leq d \leq k, \text{ e} \\ \delta'(q_i, a, q_s, 0) &= 0 && \text{, para todo } q_s \in Q - \{q_{j_i}, \cdots, q_{j_k}\} \end{aligned} \tag{4.1}$$

Desse modo, se $\delta(q_i, a) = \{q_j\}$, então $\delta'(q_i, a, q_j, 0) = 1$ pois só há um elemento no conjunto e sua probabilidade será 1 (um). Caso haja mais de um elemento no conjunto (não-determinismo), por exemplo, $\delta(q_i, a) = \{q_{j_1}, q_{j_2}\}$, tem-se que $\delta'(q_i, a, q_{j_1}, 0) = 1/\sqrt{2}$ e $\delta'(q_i, a, q_{j_2}, 0) = 1/\sqrt{2}$. Portanto, cada transição tem probabilidade $1/2$. A probabilidade será 0 (zero) quando não há transição para o estado.

Nos casos em que a transição leva o AFN para o conjunto vazio, define-se uma transição com probabilidade 1 (um) para o estado quântico $|q_n\rangle$, que funciona como um estado de "lixo" ou rejeição. O AFQ permanece no estado $|q_n\rangle$, uma vez alcançado. As transições, para esses casos, são definidas seguindo as equações a seguir.

Formalmente, para cada entrada da função $\delta(q_i, a) = \oslash$ definir:

$$|\delta(q_i, a)| = 1 \quad \text{e} \quad \delta'(q_i, a, q_n, 0) = 1 \tag{4.2}$$

e $\forall a \in \Sigma$ definir:

$$|\delta(q_n, a)| = 1 \quad \text{e} \quad \delta'(q_n, a, q_n, 0) = 1 \tag{4.3}$$

4. O estado inicial também possui seu correspondente quântico definido por $q'_{init} = |q_{init}\rangle$

5. Já os estados de aceitação do AFQ serão definidos como o espaço gerado apenas pelos estados de aceitação do AFN, formalmente dado por $Q'_{ac} = span\{|q\rangle : q \in Q_{ac}\}$, que podem ser representados pelos respectivos projetores $|q\rangle \langle q|$.

6. Deve-se construir uma matriz unitária para cada um dos símbolos do alfabeto. A partir da definição do δ', $\forall a \in \Sigma$ e $\forall q_i \in Q$, onde $0 \leq i \leq n$ definir a matriz U_a segundo a equação a seguir.

$$U_a(|q_i\rangle |0\rangle) = |q_i\rangle \sum_{d=1}^{|\delta(q_i,a)|} \delta'(q_i, a, q_{j_d}, 0) |q_{j_d}\rangle \tag{4.4}$$

As demais entradas devem ser preenchidas com vetores ortonormais arbitrários de forma que a matriz seja unitária, como sugerido por [KW97] e [AF98]. Os vetores podem ser arbitrários porque não correspondem a nenhuma transição válida no autômato e nunca serão alcançados, no entanto, são necessários para que a restrição de unitaridade seja preservada.

Apesar de não serem elementos constituintes do AFQ, as matrizes unitárias são a forma usual de representar a função de transição do autômato, por isso, sua construção é apresentada como um dos passos do algoritmo.

Uma palavra $w = a_1 a_2 \dots a_n$ é aceita pelo AFQ Ancilla quando, ao final da computação de w, a probabilidade do autômato terminar em um estado de aceitação é maior que 0 (zero). A probabilidade de aceitação $P_{ac}(w)$ é calculada pela expressão a seguir.

$$P_{ac}(w) = |Q'_{ac} U_w (|q_{init}\rangle \, |0\rangle)|^2 \begin{cases} > 0 & \text{a palavra é aceita pelo AFQ} \\ = 0 & \text{a palavra é rejeitada pelo AFQ} \end{cases} \tag{4.5}$$

onde $U_w = U_{a_1} \cdot U_{a_2} \dots U_{a_n}$

A cada aplicação das matrizes unitárias U_{a_i} que correspondem aos símbolos da palavra w, os valores do qubit que representa o estado do autômato e o qubit auxiliar são trocados entre si, e o valor do qubit auxiliar é zerado. Essa alteração é necessária para que, ao aplicar a matriz unitária, o primeiro qubit seja sempre o estado atual do autômato, e o segundo, seja o qubit auxiliar com valor 0.

4.3 Exemplo de Utilização do Algoritmo

Para ilustrar o uso do algoritmo apresentado na seção 4.2 e facilitar o entendimento, segue um exemplo do seu uso.

Seja o AFN $A = (Q, \Sigma, \delta, q_{init}, Q_{ac})$, onde $Q = \{q_0, q_1\}$, $\Sigma = \{0, 1\}$, $q_{init} = q_0$, $Q_{ac} = \{q_1\}$ e cuja função δ é representada pela tabela de transição 4.1. A representação por diagrama de estados é mostrada na figura 4.1.

Tabela 4.1: Representação do AFN A de exemplo por tabela de transição de estados

$\delta(q_0, 0) = \{q_0, q_1\}$	$\delta(q_0, 1) = \oslash$
$\delta(q_1, 0) = \oslash$	$\delta(q_1, 1) = \{q_1\}$

Seguindo a descrição do algoritmo, tem-se que $B = (Q', \Sigma, \Omega, \delta', q_{init}, Q_{ac})$ é dado por:

$Q' = \{|q_0\rangle, |q_1\rangle, |q_2\rangle\}$, $\Sigma = \{0, 1\}$, $\Omega = \{0, 1\}$, $q'_{init} = |q_0\rangle$, $Q'_{ac} = \{|01\rangle \langle 01|\}$ e δ' é então:

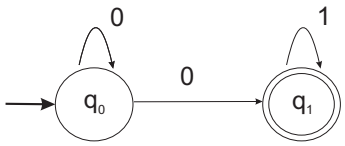


Figura 4.1: Representação do AFN A de exemplo por diagrama de estado

$$\begin{aligned}
&\delta'(|q_0\rangle, 0, |q_0\rangle, 0) = \tfrac{1}{\sqrt{2}} && \delta'(|q_0\rangle, 1, |q_0\rangle, 0) = 0\\
&\delta'(|q_0\rangle, 0, |q_1\rangle, 0) = \tfrac{1}{\sqrt{2}} && \delta'(|q_0\rangle, 1, |q_1\rangle, 0) = 0\\
&\delta'(|q_0\rangle, 0, |q_2\rangle, 0) = 0 && \delta'(|q_0\rangle, 1, |q_2\rangle, 0) = 1\\
&\delta'(|q_1\rangle, 0, |q_0\rangle, 0) = 0 && \delta'(|q_1\rangle, 1, |q_0\rangle, 0) = 0\\
&\delta'(|q_1\rangle, 0, |q_1\rangle, 0) = 0 && \delta'(|q_1\rangle, 1, |q_1\rangle, 0) = 1\\
&\delta'(|q_1\rangle, 0, |q_2\rangle, 0) = 1 && \delta'(|q_1\rangle, 1, |q_2\rangle, 0) = 0\\
&\delta'(|q_2\rangle, 0, |q_0\rangle, 0) = 0 && \delta'(|q_2\rangle, 1, |q_0\rangle, 0) = 0\\
&\delta'(|q_2\rangle, 0, |q_1\rangle, 0) = 0 && \delta'(|q_2\rangle, 1, |q_1\rangle, 0) = 0\\
&\delta'(|q_2\rangle, 0, |q_2\rangle, 0) = 1 && \delta'(|q_2\rangle, 1, |q_2\rangle, 0) = 1
\end{aligned}$$

Uma forma simplificada de escrever o δ' do AFQ B é apresentada a seguir.:

$$\begin{aligned}
&\delta'(|q_0\rangle, 0) = \tfrac{1}{\sqrt{2}}(|q_0\rangle + |q_1\rangle) && \delta'(|q_0\rangle, 1) = |q_2\rangle\\
&\delta'(|q_1\rangle, 0) = |q_2\rangle && \delta'(|q_1\rangle, 1) = |q_1\rangle\\
&\delta'(|q_2\rangle, 0) = |q_2\rangle && \delta'(|q_2\rangle, 1) = |q_2\rangle
\end{aligned}$$

Os estados $|q_0\rangle$, $|q_1\rangle$ e $|q_2\rangle$ são escritos na base computacional como $|00\rangle$, $|01\rangle$ e $|10\rangle$ respectivamente. O qubit auxiliar $|0\rangle$ da equação 4.4 transforma-se em $|00\rangle$ pois o mesmo deve possuir a mesma quantidade de qubits usados para representar os estados. Assim, para construir a matriz U_0 tem-se:

$$\begin{aligned}
&U_0(|00\rangle|00\rangle) = |00\rangle \tfrac{1}{\sqrt{2}}(|00\rangle + |01\rangle) && \text{(definida a partir de } \delta'(|q_0\rangle, 0))\\
&U_0(|01\rangle|00\rangle) = |01\rangle|10\rangle && \text{(definida a partir de } \delta'(|q_1\rangle, 0))\\
&U_0(|10\rangle|00\rangle) = |10\rangle|10\rangle && \text{(definida a partir de } \delta'(|q_2\rangle, 0))\\
&U_0(|11\rangle|00\rangle) = |11\rangle|00\rangle && \text{(definida apenas para completar a matriz)}
\end{aligned}$$

As demais entradas são calculadas apenas para completar a matriz e torná-la unitária. Os vetores devem ser ortogonais entre si e seguem abaixo:

$$\begin{aligned}
&U_0(|00\rangle|01\rangle) = |00\rangle \tfrac{1}{\sqrt{2}}(|00\rangle - |01\rangle) && U_0(|00\rangle|10\rangle) = |00\rangle \tfrac{1}{\sqrt{2}}(|10\rangle + |11\rangle) && U_0(|00\rangle|11\rangle) = |00\rangle \tfrac{1}{\sqrt{2}}(|10\rangle - |11\rangle)\\
&U_0(|01\rangle|01\rangle) = |01\rangle|11\rangle && U_0(|01\rangle|10\rangle) = |01\rangle|00\rangle && U_0(|01\rangle|11\rangle) = |01\rangle|01\rangle\\
&U_0(|10\rangle|01\rangle) = |10\rangle|11\rangle && U_0(|10\rangle|10\rangle) = |10\rangle|00\rangle && U_0(|10\rangle|11\rangle) = |10\rangle|01\rangle\\
&U_0(|11\rangle|01\rangle) = |11\rangle|01\rangle && U_0(|11\rangle|10\rangle) = |11\rangle|10\rangle && U_0(|11\rangle|11\rangle) = |11\rangle|11\rangle
\end{aligned}$$

Similarmente para U_1, tem-se:

$$U_1(|00\rangle\,|00\rangle) = |00\rangle\,|10\rangle \quad \text{(definida a partir de } \delta'(|q_0\rangle\,,1)\text{)}$$
$$U_1(|01\rangle\,|00\rangle) = |01\rangle\,|01\rangle \quad \text{(definida a partir de } \delta'(|q_1\rangle\,,1)\text{)}$$
$$U_1(|10\rangle\,|00\rangle) = |10\rangle\,|10\rangle \quad \text{(definida a partir de } \delta'(|q_2\rangle\,,1)\text{)}$$
$$U_1(|11\rangle\,|00\rangle) = |11\rangle\,|00\rangle \quad \text{(definida apenas para completar a matriz)}$$

As demais entradas seguem abaixo:

$$U_1(|00\rangle\,|01\rangle) = |00\rangle\,|11\rangle \quad U_1(|00\rangle\,|10\rangle) = |00\rangle\,|00\rangle \quad U_1(|00\rangle\,|11\rangle) = |00\rangle\,|01\rangle$$
$$U_1(|01\rangle\,|01\rangle) = |01\rangle\,|00\rangle \quad U_1(|01\rangle\,|10\rangle) = |01\rangle\,|10\rangle \quad U_1(|01\rangle\,|11\rangle) = |01\rangle\,|11\rangle$$
$$U_1(|10\rangle\,|01\rangle) = |10\rangle\,|11\rangle \quad U_1(|10\rangle\,|10\rangle) = |10\rangle\,|00\rangle \quad U_1(|10\rangle\,|11\rangle) = |10\rangle\,|01\rangle$$
$$U_1(|11\rangle\,|01\rangle) = |11\rangle\,|01\rangle \quad U_1(|11\rangle\,|10\rangle) = |11\rangle\,|10\rangle \quad U_1(|11\rangle\,|11\rangle) = |11\rangle\,|11\rangle$$

As matrizes de U_0 e U_1 seguem abaixo:

$$U_0 = \begin{bmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

$$U_1 = \begin{bmatrix}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

O cálculo da aceitação da palavra $w = 01$ pelo AFQ B recém-construído é apresentado a seguir.

$$\begin{aligned}
P_{ac}(w) &= |Q'_{ac}U_w(|q_0\rangle\,|0\rangle)|^2 \\
&= |\,|q_1\rangle\,\langle q_1|\,U_1U_0(|00\rangle\,|00\rangle)|^2 \\
&= |\,|01\rangle\,\langle 01|\,U_1(|00\rangle\,\frac{1}{\sqrt{2}}(|00\rangle + |01\rangle))|^2
\end{aligned}$$

Após a aplicação de U_0, os valores dos qubits $|00\rangle$ e $\frac{1}{\sqrt{2}}(|00\rangle + |01\rangle)$ devem ser trocados de ordem, e o segundo qubit deve receber o valor 0 (zero), para que U_1 possa ser aplicada corretamente.

$$\begin{aligned}
P_{ac}(w) &= |\,|01\rangle\,\langle 01|\,U_1(\frac{1}{\sqrt{2}}(|00\rangle + |01\rangle)\,|00\rangle)|^2 \\
&= |\,|01\rangle\,\langle 01|\,(\frac{1}{\sqrt{2}}\,|00\rangle\,|10\rangle + \frac{1}{\sqrt{2}}\,|01\rangle\,|01\rangle)|^2
\end{aligned}$$

Como não há mais nenhuma matriz a ser aplicada, não há necessidade de efetuar a troca

e zerar o segundo qubit. Pode-se então aplicar o projetor dos estados de aceitação.

$$\begin{aligned} P_{ac}(w) &= \left|\frac{1}{\sqrt{2}}\right|^2 \\ &= \frac{1}{2} \end{aligned}$$

Portanto, a palavra $w = 01$ é aceita pelo autômato quântico B, pois a probabilidade $P_{ac}(w) = \frac{1}{2}$ é maior que zero.

4.4 Corretude do Algoritmo

O AFQ Ancilla, construído de acordo com o algoritmo descrito na seção anterior, deve aceitar toda palavra que é aceita pelo AFN original, e deve rejeitar toda palavra que não é aceita pelo AFN original. Desse modo, atesta-se que o algoritmo elaborado está correto.

Para que a prova de corretude do algoritmo possa ser bem compreendida, é necessário apresentar algumas definições.

Classicamente, para simplificar a notação durante a computação de uma palavra, é utilizada uma extensão da função de transição representada por $\hat{\delta}$. Essa função de transição estendida é definida por $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$, e retorna o estado que o autômato alcança quando inicia em algum estado $q \in Q$ e computa uma seqüência de símbolos de entrada $w \in \Sigma^*$. Formalmente, tem-se:

$$\begin{aligned} \hat{\delta}(q, \epsilon) &= q \\ \hat{\delta}(q, wa) &= \delta(\hat{\delta}(q, w), a) \text{ , para cada } x \in \Sigma^* \text{ e } a \in \Sigma \end{aligned} \tag{4.6}$$

A aceitação de uma palavra por um autômato finito clássico (AFD ou AFN) também pode ser representada utilizando a função $\hat{\delta}$. Para isso, deve-se iniciar a computação de toda a palavra pelo estado inicial, aplicar a função $\hat{\delta}$ e verificar se o estado retornado pertence ao conjunto de estados de aceitação do autômato. A aceitação de uma palavra, seguindo esse método, é dada pela expressão a seguir:

$$\begin{aligned} \hat{\delta}(q_{init}, w) = \hat{\delta}(q_{init}, a_1 \cdots a_n) &= \{q_{j_1}, \cdots, q_{j_m}\} \text{ e} \\ \{q_{j_1}, \cdots, q_{j_m}\} \cap Q_{ac} &\neq \oslash \end{aligned} \tag{4.7}$$

Para a prova de corretude, será utilizada a análoga quântica à função $\hat{\delta}$, a função $\hat{\delta}'$ definida a seguir.

Definição 1. *Para* $\hat{\delta}(q_{init}, w) = \hat{\delta}(q_{init}, a_1 \cdots a_n) = \{q_{j_1}, \cdots, q_{j_m}\}$*, define-se* $\hat{\delta}'$ *como:*

$$\begin{aligned} \hat{\delta}'(q_{init}, w, q_{j_d}, 0) &= \hat{\delta}'(q_{init}, a_1 a_2 \cdots a_n, q_{j_d}, 0) \\ &= \sum_{d_1=1}^{|\delta(q_{init}, a_1)|} \cdots \sum_{d_n=1}^{|\delta(q_{j_{d_{n-1}}}, a_n)|} \frac{1}{\sqrt{|\delta(q_{init}, a_1)| \cdots |\delta(q_{j_{d_1 \cdots d_{n-1}}}, a_n)|}} \end{aligned} \tag{4.8}$$

para todo d*,* $1 \leq d \leq m$*.*

Para simplificar a apresentação da prova, considera-se também que U_{a_i}, definida pela equação 4.4, tem apenas um argumento, ou seja, $U_a |q_i\rangle = \sum_{d=1}^{|\delta(q_i,a)|} \delta'(q_i, a, q_{j_d}, 0) |q_{j_d}\rangle$. A aplicação sucessiva de matrizes U_{a_i} é calculada como segue.

Definição 2. *Para* $w = a_1 a_2 \cdots a_n$ *tem-se que* $U_w(|q_{init}\rangle) = U_{a_n} \cdots U_{a_2} U_{a_1} |q_{init}\rangle$ *é dado por:*

$$\begin{aligned} U_w(|q_{init}\rangle) &= U_{a_n} \cdots U_{a_2} U_{a_1}(|q_{init}\rangle) \\ &= \sum_{d_1=1}^{|\delta(q_{init}, a_1)|} \cdots \sum_{d_n=1}^{|\delta(q_{j_{d_{n-1}}}, a_n)|} \frac{1}{\sqrt{|\delta(q_{init}, a_1)| \cdots |\delta(q_{j_{d_1 \cdots d_{n-1}}}, a_n)}} \left| q_{j_{d_1 \cdots d_n}} \right\rangle \end{aligned} \tag{4.9}$$

Para maiores detalhes sobre a definição 2, veja o apêndice C.

Teorema 1. *Seja* A *um AFN e* B *um AFQ obtido a partir de* A *pela utilização do algoritmo descrito anteriormente. Então:*

- *Se* $w \in L(A)$ *então* $w \in L(B)$
- *Se* $w \notin L(A)$ *então* $w \notin L(B)$

Prova.

1. Para $|w| = n$, se $w \in L(A)$ então $w \in L(B)$.
   Seja $w = a_1 a_2 \cdots a_n$ e $w \in L(A)$.

   Para que w também pertença a $L(B)$, é preciso que a probabilidade de aceitação $P_{ac}(w)$ seja maior que 0 (zero), segundo a definição 4.5. Assim, precisa-se verificar se $|Q'_{ac} U_w |q_{init}\rangle|^2 > 0$.

Como $w \in L(A)$, pelo método de aceitação dado por 4.8, tem-se que: $\hat{\delta}(q_{init}, w) = \hat{\delta}(q_{init}, a_1 \cdots a_n) = \{q_{j_1}, \cdots, q_{j_m}\}$ e $\{q_{j_1}, \cdots, q_{j_m}\} \cap Q_{ac} \neq \oslash$.
Como $\{q_{j_1}, \cdots, q_{j_m}\} \cap Q_{ac} \neq \oslash$, então, a partir da expressão 4.5 e utilizando a definição 2, tem-se que:

$$\begin{aligned} P_{ac}(w) &= |Q'_{ac} U_w |q_{init}\rangle|^2 \\ &= |Q'_{ac} U_{a_n} \cdots U_{a_2} U_{a_1} |q_{init}\rangle|^2 \\ &= \left| Q'_{ac} \sum_{d_1=1}^{|\delta(q_{init}, a_1)|} \cdots \sum_{d_n=1}^{|\delta(q_{j_{d_{n-1}}}, a_n)|} \frac{1}{\sqrt{|\delta(q_{init}, a_1)| \cdots |\delta(q_{j_{d_1 \cdots d_{n-1}}}, a_n)}} \left| q_{j_{d_1 \cdots d_n}} \right\rangle \right|^2 \\ &> 0 \end{aligned} \tag{4.10}$$

A probabilidade $P_{ac}(w)$ é maior que 0 (zero) pois algum estado $\left| q_{j_{d_1 \cdots d_n}} \right\rangle$ faz parte do conjunto de estados de aceitação do AFN. Conseqüentemente, de acordo com o algoritmo, o espaço de estados de aceitação o terá como elemento gerador, resultando em $Q'_{ac} \left| q_{j_{d_1 \cdots d_n}} \right\rangle > 0$. Assim, segundo 4.5, $w \in L(B)$. □

2. Para $|w| = n$, se $w \notin L(A)$ então $w \notin L(B)$.
Seja $w = a_1 a_2 \cdots a_n$ e $w \notin L(A)$.

Para que w não pertença a $L(B)$, é preciso que a probabilidade de aceitação $P_{ac}(w)$ seja igual a 0 (zero), segundo a definição 4.5. Assim, precisa-se verificar se $|Q'_{ac} U_w |q_{init}\rangle|^2 = 0$.

Como $w \notin L(A)$, pelo método de aceitação dado por 4.8, tem-se que: $\hat{\delta}(q_{init}, w) = \hat{\delta}(q_{init}, a_1 \cdots a_n) = \{q_{j_1}, \cdots, q_{j_m}\}$ e $\{q_{j_1}, \cdots, q_{j_m}\} \cap Q_{ac} = \oslash$, isto é, nenhum estado de aceitação é alcançado a partir do estado inicial.
Como $\{q_{j_1}, \cdots, q_{j_m}\} \cap Q_{ac} = \oslash$, então, a partir da expressão 4.5 e utilizando a definição 2, tem-se que:

$$\begin{aligned} P_{ac}(w) &= |Q'_{ac} U_w |q_{init}\rangle|^2 \\ &= |Q'_{ac} U_{a_n} \cdots U_{a_2} U_{a_1} |q_{init}\rangle|^2 \\ &= \left| Q'_{ac} \sum_{d_1=1}^{|\delta(q_{init}, a_1)|} \cdots \sum_{d_n=1}^{|\delta(q_{j_{d_{n-1}}}, a_n)|} \frac{1}{\sqrt{|\delta(q_{init}, a_1)| \cdots |\delta(q_{j_{d_1 \cdots d_{n-1}}}, a_n)}} \left| q_{j_{d_1 \cdots d_n}} \right\rangle \right|^2 \\ &= 0 \end{aligned} \tag{4.11}$$

A probabilidade $P_{ac}(w)$ é igual a 0 (zero) pois nenhum estado $\left| q_{j_{d_1 \cdots d_n}} \right\rangle$ faz parte do conjunto de estados de aceitação do AFN. Conseqüentemente, de acordo com o algoritmo, o espaço de estados de aceitação não o terá como elemento gerador, resultando em $Q'_{ac} \left| q_{j_{d_1 \cdots d_n}} \right\rangle = 0$. Assim, segundo 4.5, $w \notin L(B)$. □

4.5 Terceira Etapa: Circuito Quântico para Ancilla AFQ

Um AFQ Ancilla, construído segundo o algoritmo apresentado, pode ser implementado com pouco esforço em um circuito quântico. Para aqueles não familiarizados com os circuitos quânticos, sugere-se a leitura do apêndice B. A figura 4.2 abaixo mostra o arcabouço ou circuito genérico para um AFQ Ancilla qualquer. Nessa abordagem, o alfabeto dos autômatos é $\Sigma = \{0, 1\}$.

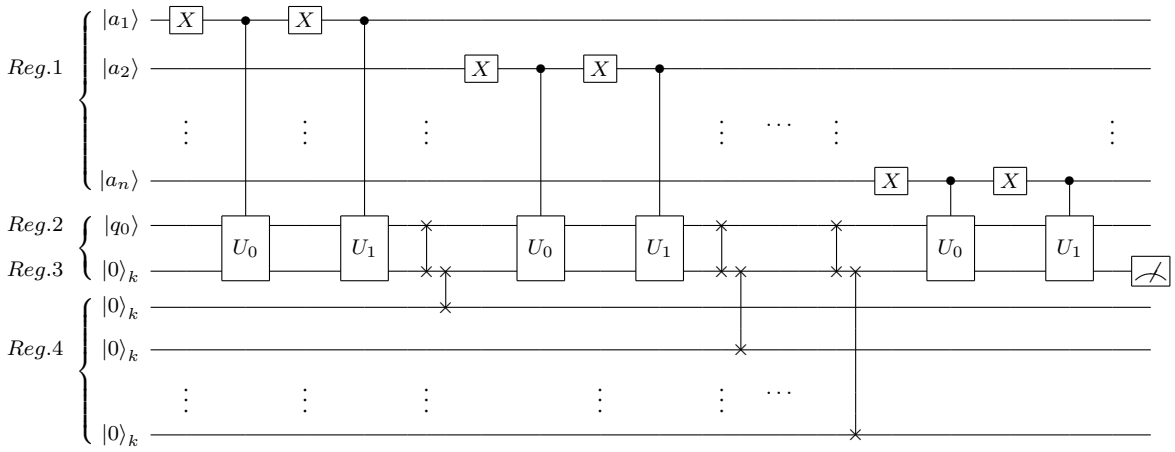


Figura 4.2: Circuito quântico que implementa um AFQ *ancilla*.

A palavra a ser computada, $w \in \{0, 1\}^n$, é representada pelo registrador 1, composto pelos n primeiros qubits do circuito, onde o valor de cada qubit corresponde a um dos símbolos da palavra.

O estado atual do autômato é representado pelo registrador 2, composto por k qubits. Os k qubits correspondem à quantidade de bits necessários para escrever em binário os m estados do autômato. Logo, $k = \lceil \log m \rceil$. Esse registrador deve ser iniciado com o valor do estado inicial do autômato, $|q_0\rangle$.

O qubit ancilla é representado pelo registrador 3, também composto por k qubits. Seu valor inicial é $|0\rangle_k$ e sua função é armazenar temporariamente o valor do próximo estado do autômato. Após a computação de cada símbolo da palavra, seu valor deve ser transferido para o registrador 2 atualizando o estado atual do autômato, e em seguida, re-estabelecido para $|0\rangle_k$. Essa mudança de valores entre os registradores é feita pelas portas $SWAP$ presentes no circuito.

Os demais qubits do registrador 4 são apenas auxiliares usados para re-estabelecer o valor do qubit ancilla para $|0\rangle_k$. Cada qubit do registrador 4 é iniciado com o valor $|0\rangle_k$.

A função de transição (δ) do AFQ é representada pelas portas U_0 e U_1, correspondentes às matrizes definidas na segunda fase da abordagem. A aplicação de cada porta aos registradores 2 e 3 é controlada pelo símbolo lido: U_0 é aplicada quando $a_i = 0$ e U_1, quando $a_i = 1$. As portas X são utilizadas apenas para estabelecer o controle para $a_i = 0$.

O estado final do autômato é representado pela porta de medição, aplicada ao qubit ancilla, registrador 3, após a computação de todos os símbolos da palavra de entrada.

A execução do circuito é realizada da esquerda para a direita, aplicando seqüencialmente as portas aos qubits, até que, ao final do circuito, o qubit ancilla é medido. A palavra é aceita, se a probabilidade resultante da medição for maior que zero, caso contrário a palavra é rejeitada.

Para ilustrar a codificação e execução de um AFQ ancilla, segue abaixo, na figura 4.3 o circuito correspondente ao AFQ ancilla B construído no exemplo da seção anterior, com a palavra de entrada $w = 001$.

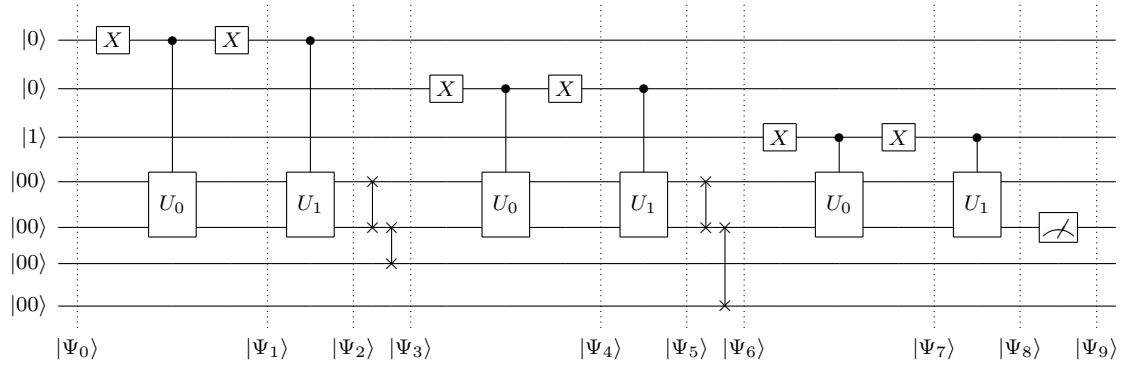


Figura 4.3: Circuito quântico que implementa o AFQ *ancilla* B com $w = 001$.

Para facilitar o entendimento do passo-a-passo da execução do circuito, foram acrescentadas as indicações sobre o estado do sistema em diferentes intervalos de tempo, representadas pelos $|\psi_i\rangle$.

A execução do circuito segue abaixo:

$|\psi_0\rangle = |0\rangle\,|0\rangle\,|1\rangle\,|00\rangle\,|00\rangle\,|00\rangle\,|00\rangle$

$|\psi_1\rangle = |0\rangle\,|0\rangle\,|1\rangle\,|00\rangle\left[\frac{1}{\sqrt{2}}(|00\rangle + |01\rangle)\right]|00\rangle\,|00\rangle$

$$|\psi_2\rangle = |0\rangle\,|0\rangle\,|1\rangle\,|00\rangle\left[\tfrac{1}{\sqrt{2}}(|00\rangle + |01\rangle)\right]|00\rangle\,|00\rangle$$
$$|\psi_3\rangle = |0\rangle\,|0\rangle\,|1\rangle\left[\tfrac{1}{\sqrt{2}}(|00\rangle + |01\rangle)\right]|00\rangle\,|00\rangle\,|00\rangle$$
$$|\psi_4\rangle = |0\rangle\,|0\rangle\,|1\rangle\left[\tfrac{1}{\sqrt{2}}(|00\rangle + |01\rangle)\right]\left[\tfrac{1}{2}(|00\rangle + |01\rangle) + \tfrac{1}{\sqrt{2}}\,|10\rangle\right]|00\rangle\,|00\rangle$$
$$|\psi_5\rangle = |0\rangle\,|0\rangle\,|1\rangle\left[\tfrac{1}{\sqrt{2}}(|00\rangle + |01\rangle)\right]\left[\tfrac{1}{2}(|00\rangle + |01\rangle) + \tfrac{1}{\sqrt{2}}\,|10\rangle\right]|00\rangle\,|00\rangle$$
$$|\psi_6\rangle = |0\rangle\,|0\rangle\,|1\rangle\left[\tfrac{1}{2}(|00\rangle + |01\rangle) + \tfrac{1}{\sqrt{2}}\,|10\rangle\right]|00\rangle\,|00\rangle\left[\tfrac{1}{\sqrt{2}}(|00\rangle + |01\rangle)\right]$$
$$|\psi_7\rangle = |0\rangle\,|0\rangle\,|1\rangle\left[\tfrac{1}{2}(|00\rangle + |01\rangle) + \tfrac{1}{\sqrt{2}}\,|10\rangle\right]|00\rangle\,|00\rangle\left[\tfrac{1}{\sqrt{2}}(|00\rangle + |01\rangle)\right]$$
$$|\psi_8\rangle = |0\rangle\,|0\rangle\,|1\rangle\left[\tfrac{1}{2}(|00\rangle + |01\rangle) + \tfrac{1}{\sqrt{2}}\,|10\rangle\right]\left[\tfrac{1}{2}(|10\rangle + |01\rangle) + \tfrac{1}{\sqrt{2}}\,|10\rangle\right]|00\rangle\left[\tfrac{1}{\sqrt{2}}(|00\rangle + |01\rangle)\right]$$
$$|\psi_9\rangle = |0\rangle\,|0\rangle\,|1\rangle\left[\tfrac{1}{2}(|00\rangle + |01\rangle) + \tfrac{1}{\sqrt{2}}\,|10\rangle\right]\tfrac{1}{2}\,|01\rangle\,|00\rangle\left[\tfrac{1}{\sqrt{2}}(|00\rangle + |01\rangle)\right]$$

Como $|01\rangle$ representa o estado de aceitação q_2 e sua a amplitude é $\frac{1}{2}$, o AFQ B aceita a palavra $w = 001$ com probabilidade $\left(\frac{1}{2}\right)^2 = \frac{1}{4}$.

Para diminuir a complexidade na implementação de portas que operam sobre múltiplos qubits, é comum decompô-las em portas quânticas mais simples mas que executam a mesma operação. As portas U_0 e U_1 do exemplo sendo estudado foram decompostas em portas multi-controladas e seus respectivos circuitos são apresentados nas figuras 4.4 e 4.5, respectivamente. Detalhes sobre métodos de decomposição podem ser encontrados em [NC00].

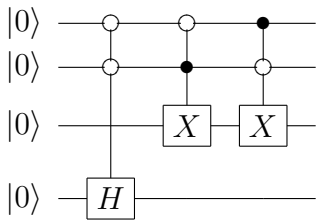


Figura 4.4: Circuito quântico que implementa a porta U_0.

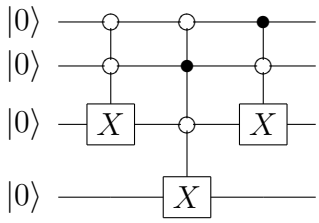


Figura 4.5: Circuito quântico que implementa a porta U_1.

4.6 Análise da Abordagem

4.6.1 Análise da Primeira Etapa

O algoritmo clássico ER-AFN utilizado na etapa inicial da abordagem e apresentado na seção 2.5.1, possui complexidade $O(m)$ para converter uma expressão regular de tamanho m em um autômato finito não-determinístico com transições ϵ. Para eliminar as transições ϵ de um ϵ-AFN com n estados leva-se tempo $O(n^3)$, sem aumentar o número de estados [HMU02]. Sendo assim, a primeira etapa da abordagem mostra-se computacionalmente eficiente.

4.6.2 Análise da Segunda Etapa

Durante a segunda etapa, é realizada a transformação do AFN em um AFQ Ancilla utilizando o algoritmo desenvolvido e apresentado na seção 4.2. A análise de complexidade do algoritmo é feita avaliando o tempo gasto em cada um dos passos de definição dos elementos do AFQ Ancilla. Os passos que definem Σ, Ω, q'_{init} e Q'_{ac} são triviais e demandam tempo $O(1)$ cada. A definição de Q' consiste numa iteração sobre os n estados do AFN definindo os estados correspondentes no AFQ Ancilla e, portanto, demanda tempo $O(n)$. Para construir o δ' também será gasto tempo linear proporcional à quantidade de transições do autômato.

O passo que demandará mais tempo, mas ainda assim com complexidade polinomial, é a definição das matrizes U_a pois as mesmas possuem dimensão $2^{2\cdot\lceil\log n\rceil} \times 2^{2\cdot\lceil\log n\rceil}$. No entanto só é necessário definir os valores das $2^{2\cdot\lceil\log n\rceil}$ colunas da matriz com vetores ortonormais entre si, calculados a partir dos n vetores já definidos pelo δ' do AFQ. Logo, essa etapa possui complexidade da ordem de $O(2^{2\cdot\lceil\log n\rceil})$, que é quadrática em relação ao número de estados do autômato, pois de acordo com as propriedades das funções logarítmica e exponencial, tem-se que $2^{2\cdot\log n} = 2^{\log n\cdot 2} = (2^{\log n})^2 = n^2$. Desse modo, a complexidade do algoritmo desenvolvido é da ordem de $O(n^2)$ e a segunda etapa mostra-se, também, computacionalmente eficiente.

4.6.3 Análise da Terceira Etapa

O circuito quântico proposto na terceira etapa da abordagem deve ser analisado sob três aspectos: quantidade de qubits utilizados, quantidade de portas utilizadas e tempo de execução

na computação de uma palavra de tamanho p para um autômato de n estados.

A quantidade de qubits utilizados no circuito é computada como segue: para representar a palavra de entrada são utilizados p qubits, um para cada símbolo da palavra; para representar o estado atual e o qubit ancilla são utilizados $2k = 2 \cdot \lceil \log n \rceil$ qubits; e para representar o registrador 4 auxiliar são necessários $(p-1) \cdot k$ qubits. Logo, a quantidade de qubits utilizadas no circuito é dada pela equação 4.12:

$$p + 2k + (p-1)k = p + k + pk \tag{4.12}$$

cuja complexidade é $O(pk) = O(p \cdot \lceil \log n \rceil)$, sendo linearmente proporcional à quantidade de símbolos da palavra e ao logaritmo da quantidade de estados do autômato.

A contagem das portas utilizadas pode ser feita analisando as portas de múltiplos qubits como oráculos ou caixas-pretas, de modo que para cada porta é contada apenas uma unidade, ou pode ser feita decompondo tais portas em portas básicas e fazendo o cálculo a partir dessas novas variáveis.

O cálculo com oráculos é trivial e é apresentado a seguir: para cada um dos p símbolos da palavra são utilizadas duas portas X e as duas portas U_0 e U_1; para $(p-1)$ símbolos são utilizadas duas portas $SWAP$ e, por fim, uma porta de medição M. A quantidade total de portas é dada pela equação 4.13:

$$p \cdot (2X + 2U) + (p-1) \cdot 2k \cdot SWAP + M = 2X \cdot p + 2U \cdot p + 2k \cdot SWAP \cdot p - 2k \cdot SWAP + M \tag{4.13}$$

Logo a quantidade de portas é $O(pk)$, linearmente proporcional à quantidade de símbolos da palavra e ao logaritmo da quantidade de estados do autômato.

Se o circuito for analisado apenas com portas básicas é necessário decompor todas as portas sobre múltiplos qubits em portas quânticas básicas, tais como as portas $CNOT$ e sobre um qubit H e X. Uma porta bem simples que age em um único qubit apenas também pode ser decomposta, como é o caso das portas $SWAP$ que são implementadas utilizando três portas $CNOT$. No entanto, o método de decomposição descrito por Nielsen e Chuang [NC00] pode ser utilizado para decompor qualquer matriz U arbitrária que age sobre múltiplos qubits. Inicialmente a matriz U deve ser decomposta em matrizes de dois níveis, isto é, aquelas que agem não trivialmente apenas em dois ou menos componentes do espaço. Para uma matriz U de dimensão $d \otimes d$ são utilizadas $r \leq d(d-1)/2 = O(d^2)$ matrizes de dois níveis, onde

$d = 2^{2k}$. Cada uma dessas matrizes pode ser então implementada em circuito, através da técnica descrita em [NC00], utilizando $O((2k)^2)$ portas sobre um qubit e portas CNOT. Todo o processo de decomposição demandará então $O((2^{2k})^2 \cdot (2k)^2) = O(16^k \cdot 4k^2)$ portas básicas e será, portanto, polinomial ao número de estados do autômato, pois, como $k = \lceil \log n \rceil$, tem-se que $16^k \cdot 4k^2 = (2^4)^k \cdot 4k^2 = (2^k)^4 \cdot 4k^2 = (2^{\log n})^4 \cdot 4(\log n)^2 = n^4 \cdot 4(\log n)^2$.

No exemplo apresentado na seção anterior, as portas U_0 e U_1 foram implementadas por portas com três e quatro controles de modo que ainda é possível decompô-las em portas mais básicas utilizando o método de Nielsen e Chuang [NC00].

Finalizando a análise, o tempo gasto para executar a computação de uma palavra no circuito é diretamente proporcional à quantidade de portas do mesmo, e portanto, a execução é de ordem polinomial.

Capítulo 5

Conclusão

Neste capítulo são apresentadas as considerações finais desta dissertação e são sugeridos alguns trabalhos futuros que visam contribuir com a solução apresentada neste documento.

5.1 Conclusões

Esse trabalho de dissertação visou solucionar o problema de ineficiência das abordagens clássicas existentes no uso de expressões regulares. Como solução, foi proposta uma abordagem alternativa fazendo uso de elementos da Computação Quântica para tornar o processo de utilização de expressões regulares eficiente. A abordagem proposta consiste em três etapas: i) transformação ER-AFN, ii) transformação AFN - AFQ Ancilla e iii) implementação do AFQ Ancilla em circuito quântico. A etapa (i) é clássica e consiste na utilização do algoritmo de transformação de uma expressão regular em um autômato finito não-determinístico reconhecidamente eficiente. Para as demais etapas foi necessário a escolha de um modelo de autômato finito quântico adequado (AFQ Ancilla), que reconhece no mínimo a classe das linguagens regulares, tendo sido então desenvolvidos um algoritmo de transformação de AFN para AFQ Ancilla e um arcabouço para implementação do AFQ Ancilla em circuito quântico.

O algoritmo de transformação AFN-AFQ Ancilla é de simples execução e pode ser utilizado para transformar qualquer AFN em um AFQ Ancilla que reconhece a mesma linguagem, como provado em 4.4. Com relação à eficiência, foram analisados a quantidade de estados do autômato quântico resultante e o tempo de execução do algoritmo. A quantidade

de estados do AFQ é a mesma do AFN, podendo ter apenas um estado a mais, caso alguma transição do AFN esteja ausente. Essa linearidade é a grande vantagem em relação às abordagens clássicas pois, no pior caso, havia um aumento exponencial no número de estados na transformação AFN-AFD e necessitava-se ainda minimizar o autômato. O algoritmo desenvolvido também é eficiente em relação ao tempo de execução, pois o mesmo executa em tempo polinomial ao número de estados do autômato.

O arcabouço para implementar o AFQ Ancilla na linguagem de circuitos quânticos também apresenta vantagens devido, principalmente, à sua flexibilidade e modularização, pois, para implementar um autômato qualquer, basta definir o número de qubits e construir as portas unitárias que codificam o seu δ. O circuito apresenta um conjunto fixo de portas utilizadas na leitura de um símbolo, de modo que a troca da palavra de entrada modifica apenas a quantidade de módulos utilizados.

O método apresentado leva em consideração apenas o alfabeto binário $\Sigma = \{0, 1\}$ mas é possível utilizar alfabetos com mais de dois símbolos. Seria necessário representá-los em notação binária, de forma que cada símbolo seria representado por $\lceil \log m \rceil$ qubits, onde $m = |\Sigma|$. Outra implicação seria o aumento da quantidade de matrizes U_i no circuito, já que é preciso definir uma matriz para cada símbolo do alfabeto. Para escolher qual matriz aplicar ao ler um símbolo da palavra, são necessário circuitos mais elaborados, como mostra o exemplo abaixo.

Para $\Sigma = |0, 1, 2, 3, 4|$, $m = 5$, tem-se $|a_i\rangle = |a_1 a_2 a_3\rangle$. Assim: $|0\rangle = |000\rangle$, $|1\rangle = |001\rangle$, $|2\rangle = |010\rangle$, $|3\rangle = |011\rangle$ e $|4\rangle = |100\rangle$.

No circuito, o controle será fechado para os bits $a_i = 1$, e será aberto para os bits $a_i = 0$ (veja o Apêndice A). Parte do circuito que ilustra a utilização das matrizes na leitura de um símbolo da palavra segue na figura 5.1:

5.2 Trabalhos Futuros

Como apresentado na seção 4.2, um dos passos do algoritmo AFN-AFQ Ancilla é a definição das matrizes unitárias U_i, que são de dimensão polinomialmente proporcional ao número de estados. Entretanto, a maior parte das entradas das matrizes são calculadas apenas para tornar a matriz unitária, não sendo utilizadas para representar a função de transição (δ) do autômato.

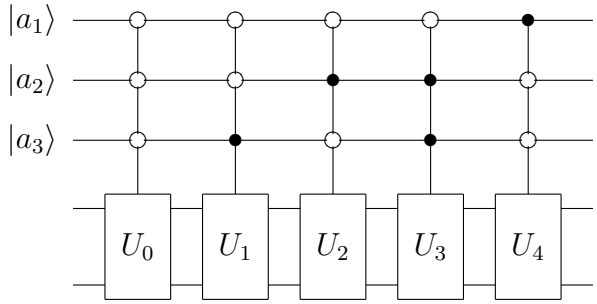


Figura 5.1: Módulo para leitura de um símbolo com alfabeto não-binário

Como conseqüência da dimensão das matrizes, a implementação das mesmas demandará um grande número de portas quânticas básicas, comprometendo a eficiência da execução do circuito correspondente. Paschen [Pas00] sugeriu ser possível escrever as matrizes com $|\Omega||Q|$ colunas e $|Q|$ linhas, reduzindo as dimensões das mesmas. Desse modo, sugere-se um estudo mais aprofundado para otimizar a representação da função de transição do autômato através de matrizes mais compactas.

Ainda nessa direção, sugere-se uma pesquisa maior sobre matrizes que atuam em múltiplos qubits e seu impacto na complexidade dos circuitos quânticos. Em geral, os algoritmos quânticos utilizam oráculos que atuam sobre múltiplos qubits, mas sua implementação real muitas vezes não é discutida. O arcabouço apresentado nesse trabalho também assume a implementação das portas U_0 e U_1 como sendo eficientes, mas devido às suas dimensões, viu-se que a quantidade de portas básicas para implementá-las pode se tornar impraticável. Sugere-se também investigar se as matrizes que implementam o δ de autômatos finitos quânticos possuem algum padrão de formação e, posteriormente, desenvolver uma técnica mais específica de decomposição de matrizes otimizando o uso das portas básicas.

Por fim, sugere-se a extensão da pesquisa na área da Teoria dos Autômatos para abranger também as gramáticas regulares e os algoritmos relacionados, investigando novas áreas onde os autômatos finitos quânticos possam ser utilizados na solução de problemas.

Bibliografia

[ABF+99] Andris Ambainis, Richard F. Bonner, Rusins Freivalds, Marats Golovkins, and Marek Karpinski. Quantum finite multitape automata. In *SOFSEM '99, Theory and Practice of Informatics, 26th Conference on Current Trends in Theory and Practice of Informatics*, volume 1725 of *Lecture Notes in Computer Science*, pages 340–348. Springer, 1999.

[ABFK99] Andris Ambainis, Richard Bonner, Rusins Freivalds, and Arnolds Kikusts. Probabilities to accept languages by quantum finite automata. In *Computing and Combinatorics, 5th Annual International Conference, COCOON'99, Lecture Notes in Computer Science*, volume 1627, pages 174–183, 1999.

[AF98] Andris Ambainis and Rusins Freivalds. 1-way quantum finite automata: strengths, weaknesses and generalizations. In *39th Annual Symposium on Foundations of Computer Science, FOCS'98*, pages 332–341. IEEE Computer Society, 1998.

[AI99] Masami Amano and Kazuo Iwama. Undecidability on quantum finite automata. In *STOC'99: Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 368–375, 1999.

[AK01] Andris Ambainis and Arnolds Kikusts. Exact results for accepting probabilities of quantum automata. In *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001*, volume 2136, pages 135–147. Springer, 2001.

[AKV01] Andris Ambainis, Arnolds Kikusts, and Maris Valdats. On the class of languages recognizable by 1-way quantum finite automata. In *STACS 01: Proceedings of*

the 18th Annual Symposium on Theoretical Aspects of Computer Science, pages 75–86. Springer-Verlag, 2001.

[Arb87] Michael A. Arbib. *Brains, machines and mathematics*. Springer-Verlag, 1987.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: principles, techniques and tools*. Addison-Wesley, 1986.

[AW02] Andris Ambainis and John Watrous. Two-way finite automata with quantum and classical states. *Theoretical Computer Science*, 287:299–311, 2002.

[BBC+95] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457–3467, 1995.

[BFR02] Richard Bonner, Rusins Freivalds, and Zigmars Rasscevskis. Size of finite probabilistic and quantum automata. available at http://citeseer.ist.psu.edu/bonner02size.html, 2002.

[BMP03] Alberto Bertoni, Carlo Mereghetti, and Beatrice Palano. Quantum computing: 1-way quantum automata. In *Developments in Language Theory, 7th International Conference, DLT 2003*, pages 1–20, 2003.

[BP99] Alex Brodsky and Nicholas Pippenger. Characterizations of 1-way quantum finite automata. Technical Report TR-99-03, University of British Columbia, UBC CS, 2 1999.

[BRSM04] Daniela Berardi, Fabio De Rosa, Luca De Santis, and Massimo Mecella. Finite state automata as conceptual model for e-services. *Journal of Integrated Design and Process Science*, 8(2):105–121, 2004.

[BV97] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.

[CH97] Maxime Crochemore and Christophe Hancart. Automata for matching patterns. *Handbook of formal languages*, 2:399–462, 1997.

[Cia01] Massimo Pica Ciamarra. Quantum reversibility and a new model of quantum automaton. In *Fundamentals of Computation Theory, 13th International Symposium, FCT 2001*, pages 376–379, 2001.

[CL89] John Carroll and Darrell Long. *Theory of finite automata with an introduction to formal languages*. Prentice-Hall International, 1989.

[Coh97] Daniel I. A. Cohen. *Introduction to computer theory*. John Wiley & Sons, Inc., second edition, 1997.

[Deu85] David Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[Deu89] David Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 425(1868):73–90, 1989.

[Fey82] Richard Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6 & 7):467–488, 1982.

[Fri02] Jeffrey E. F. Friedl. *Mastering Regular Expressions*. O'Reilly Media, second edition, 2002.

[Goy07] Jan Goyvaerts. Specialized tools and utilities for working with regular expressions. artigo on-line disponível em http://www.regular-expressions.info/tools.html, 2007.

[Gri73] David Gries. Describing an algorithm by hopcroft. *Acta Informatica*, 2:97–109, 1973.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[Hae03] Fritz von Haeseler. *Automatic Sequences*. Walter de Gruyter, Berlin, 2003.

[HMU02] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introdução à teoria dos autômatos, linguagens e computação*. Editora Campus, Rio de Janeiro, 2002.

[Hol00] Jan Holub. *Simulation of Nondeterministic Finite Automata in Pattern Matching*. PhD thesis, Faculty of Electrical Engineering, Czech Technical Univeristy, Prague, 2000.

[Hop71] John E. Hopcroft. An n log n algorithm for minimizing the states in a finite automaton. *The Theory of Machines and Computations*, pages 189–196, 1971.

[HSW01] Juraj Hromkovi, Sebastian Seiberta, and Thomas Wilke. Translating regular expressions into small epsilon-free nondeterministic finite automata. *Journal of Computer and System Sciences*, 62(4):565–588, 2001.

[Huf54] David A. Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, 257:164–190, 1954.

[IL06] Cheyenne Ribeiro Guedes Isidro and Bernardo Lula Jr. Um algoritmo para transformar um autômato finito não-determinístico em um autômato finito quântico preservando o número de estados e a linguagem reconhecida. In *Anais do Workshop-Escola de Computação e Informação Quântica - WECIQ2006*, pages 161–168, 2006.

[Kar00] Lauri Karttunen. Applications of finite-state transducers in natural language processing. In *Implementation and Application of Automata, 5th International Conference, CIAA*, pages 34–46, 2000.

[KFPL07] Luis Kowada, Celina Figueiredo, Renato Portugal, and Carlile Lavor. Aplicação do algoritmo de grover para problemas np-completos. In *2° Workshop-Escola de Computação e Informação Quântica*, 2007.

[Kik98] Arnolds Kikusts. A small 1-way quantum finite automata, 1998. quant-ph/9810065.

[Kle56] Stephen C. Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, pages 3–42, 1956.

[KLS98] Tomasz Kowaltowski, Cláudio L. Lucchesi, and Jorge Stolfi. Finite automata and efficient lexicon implementation. Technical Report IC-98-02, Institute of Computing, Universidade de Campinas, 1998.

[Koz97] Dexter C. Kozen. *Automata and computability*. Springer-Verlag, 1997.

[KW97] Attila Kondacs and John Watrous. On the power of quantum finite state automata. In *38th Annual Symposium on Foundations of Computer Science, FOCS'97*, pages 66–75. IEEE Computer Society, 1997.

[LP81] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the theory of computation*. Prentice Hall, 1981.

[Mar06] Franklin de Lima Marquezino. A transformada de fourier quântica aproximada e sua simulação. Master's thesis, Laboratório Nacional de Computação Científica, 2006.

[MC97] Cristopher Moore and James P. Crutchfield. Quantum automata and quantum grammars. *Theoretical Computer Science*, 1997.

[Mea55] George H. Mealy. A method for synthesizing sequential circuits. *Bell Systems Technical Journal*, 34:1045–1079, 1955.

[Mid02] Gatis Midrijanis. The complexity of probabilistic versus quantum finite automata. In *SOFSEM 2002: Theory and Practice of Informatics, 29th Conference on Current Trends in Theory and Practice of Informatics*, pages 273–278, 2002.

[Moh96] Mehryar Mohri. On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, 2(1):61–80, 1996.

[Moo56] Edward F. Moore. Gedanken experiments on sequential machines. *Automata Studies*, pages 129–156, 1956.

[MP43] Warren McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 7:115–133, 1943.

[Nav04] Gonzalo Navarro. Patttern matching. *Journal of Applied Statistics*, 31:925–949, 2004.

[NC00] Michael A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, New York, NY, USA, 2000.

[NIHK02] Masaki Nakanishi, Takao Indoh, Kiyoharu Hamaguchi, and Toshinobu Kashiwabara. On the power of non-deterministic quantum finite automata. *IEICE Transactions on Information and Systems*, E85-D(2):327–332, 2002.

[Nik07] Nikolay Y. Nikolaevshed. Converting nfa to dfa. class notes of CIS 324: Language Design and Implementation available at http://homepages.gold.ac.uk/nikolaev, 2007.

[OA07] Nigini A. Oliveira and Francisco M. de Assis. Algoritmo quântico para cálculo da distância mínima de códigos de bloco lineares. In *2° Workshop-Escola de Computação e Informação Quântica*, 2007.

[ONA07] Nigini A. Oliveira, Edmar Nascimento, and Francisco M. de Assis. Ataques quânticos ao gerador pseudo-aleatório de blum-micali. In *Simpósio Brasileiro de Telecomunicações - SBRT*, 2007.

[Pas00] Kathrin Paschen. Quantum finite automata using ancilla qubits. Technical report, University of Karlsruhe, May 2000.

[Per93] Dominique Perrin. Les debuts de la theorie des automates. Technical Report 1993-047, Laboratoire d'Informatique Algorithmique: Fondements et Applications, 1993.

[Pet99] Charles Petzold. *Codes*. Microsof Press, 1999.

[Ron95] Dana Ron. *Automata Learning and its Applications*. PhD thesis, Hebrew University, 1995.

[RS59] Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(4):114–125, 1959.

[RS97] Emmanuel Roche and Yves Schabes. *Finite-state language processing*. The MIT Press, 1997.

[Sal69] Arto Salomaa. *Theory of automata*. Pergamon Press, 1969.

[Sch07] Thomas Schwentick. Automata for xml—a survey. *Journal of Computer and System Sciences*, 73(3):289–315, 2007.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *IEEE Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[Sim94a] David R. Simon. On the power of quantum computation. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 116–123, Los Alamitos, CA, 1994. Institute of Electrical and Electronic Engineers Computer Society Press.

[Sim94b] Charles C. Sims. *Computation with finitely presented groups*. Cambridge University Press, 1994.

[Sip97] Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[Ski97] Steven S. Skiena. *The Algorithm Design Manual*. Springer-Verlag, New York, 1997.

[Tho68] Ken Thompson. Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

[Tur36] Alan Turing. On computable numbers, with an application to the entscheidungsproblem. In *Proceedings of the London Mathematical Society, Series 2*, pages 230–267, 1936.

[Val00] Maris Valdats. The class of languages recognizable by 1-way quantum finite automata is not closed under union. In *Proceedings of International Workshop on Quantum Computation and Learning*, pages 52–64, 2000.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *First Symposium on Logic in Computer Science*, pages 322–331, 1986.

Apêndice A

Computação Quântica: Conceitos Básicos

A.1 Representação da Informação

Na Computação Quântica (CQ), o *qubit* (*quantum bit*) é a unidade básica de informação, em analogia ao conceito clássico de *bit*. Diferentemente do bit da Computação Clássica que pode estar em apenas dois estados distintos, 0 ou 1, um *qubit*, além dos estados $|0\rangle$ e $|1\rangle$, pode estar em uma *superposição* desses dois estados na forma:

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

em que α e β são coeficientes complexos que representam a amplitude dos estados $|0\rangle$ e $|1\rangle$, respectivamente, e devem obedecer à equação:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{A.2}$$

Uma particularidade da Mecânica Quântica é que nem toda informação contida no qubit pode ser extraída. A extração de informação se dá através da medição do qubit e durante esse processo parte da informação é perdida. Se o qubit está num estado de superposição, ao medí-lo, ele colapsa para um dos estados clássicos, $|0\rangle$ ou $|1\rangle$. O módulo da amplitude de um estado quântico ao quadrado informa a probabilidade de obter esse estado ao efetuar uma medição. Por exemplo, ao medir o estado $|\psi\rangle$ da equação A.1, obtém-se o valor $|0\rangle$ com probabilidade α^2 e o valor $|1\rangle$ com probabilidade β^2.

A CQ utiliza a notação de Dirac (*bra* $\langle\,|$ e *ket* $|\,\rangle$) para representar os estados quânticos. As notações $|0\rangle$ e $|1\rangle$ representam, respectivamente, os vetores:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{A.3}$$

e a notação $|\psi\rangle$ representa o vetor:

$$|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{A.4}$$

O bra representa o transposto conjugado[1] de um *qubit* ($\langle\psi| = |\psi\rangle^{\dagger}$) e pode ser definido por:

$$\begin{aligned} \langle\psi| &= |\psi\rangle^{\dagger} \\ &= \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \\ &= \begin{bmatrix} \alpha* & \beta* \end{bmatrix} \end{aligned} \tag{A.5}$$

Em um computador clássico, os bits são agrupados em conjuntos chamados registradores. Se um bit pode armazenar um dos dois números 0 ou 1, então um registrador de n bits pode armazenar 2^n números diferentes, um por vez. No computador quântico, no entanto, um registrador de n qubits pode armazenar 2^n valores diferentes ao mesmo tempo, devido à característica da superposição. Para um sistema de mais de um qubit utiliza-se a operação de produto tensorial $\otimes$ para operar os qubits. A notação utilizada é $|a\rangle \otimes |b\rangle = |a\rangle\,|b\rangle = |ab\rangle$. O estado geral de um sistema de n qubits é apresentado na expressão A.6 abaixo:

$$|\psi\rangle = \sum_{x=0}^{2^n-1} \alpha_x\,|x\rangle\,, \tag{A.6}$$

com a restrição $\sum_{x=0}^{2^n-1} |\alpha_x|^2 = 1$ e a base computacional $\{|0\rangle\,, |1\rangle\,, \ldots, |2^n-1\rangle\}$.

$|\psi\rangle$ é descrito por um vetor unitário com amplitudes α_i, $0 \leq i \leq 2^n - 1$.

Abaixo, segue um exemplo de um sistema com dois qubits:

[1] O conjugado de um número complexo $c = a \pm bi$ é $c* = a \mp bi$

$$
\begin{aligned}
|\psi\rangle &= |\psi_1\rangle \otimes |\psi_2\rangle \\
&= (\alpha_1 |0\rangle + \beta_1 |1\rangle) \otimes (\alpha_2 |0\rangle + \beta_2 |1\rangle) \\
&= \alpha_1\alpha_2 |00\rangle + \alpha_1\beta_2 |01\rangle + \alpha_2\beta_1 |10\rangle + \beta_1\beta_2 |11\rangle \\
&= \alpha_{00} |00\rangle + \alpha_{01} |01\rangle + \alpha_{10} |10\rangle + \alpha_{11} |11\rangle \\
&= \alpha_0 |0\rangle + \alpha_1 |1\rangle + \alpha_2 |2\rangle + \alpha_3 |3\rangle \\
&= \sum_{x=0}^{3} \alpha_x |x\rangle
\end{aligned}
\tag{A.7}
$$

O resultado dessa operação é o estado quântico $|\psi\rangle$ de dois qubits que contém os estados 00, 01, 10 e 11 ao mesmo tempo, onde cada estado tem sua respectiva amplitude α_x.

A.2 Medição

De acordo com a Física Quântica, a única forma de se obter acesso à informação contida no estado $|\psi\rangle$ é através de uma medição. De uma maneira simplificada, uma medição corresponde a uma projeção do vetor de estado representando o qubit em um dos estados da base computacional. O processo de medição altera estado do sistema ψ descrito em A.1, fazendo-o assumir o estado $|0\rangle$, com probabilidade $|\alpha|^2$, ou o estado $|1\rangle$, com probabilidade $|\beta|^2$.

Então, se o qubit está no estado $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$, a probabilidade de uma medição encontrá-lo no estado $|x\rangle$, onde $x \in \{0, 1\}$, é dada por:

$$
p_\psi(x) = |\langle x|\psi\rangle|^2 \tag{A.8}
$$

Por exemplo, para $x = 0$ e $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$ tem-se:

$$
\begin{aligned}
p_\psi(0) &= |\langle 0|\psi\rangle|^2 = |\langle 0| (\alpha_0 |0\rangle + \alpha_1 |1\rangle)|^2 \\
&= |\langle 0| \alpha_0 |0\rangle + \langle 0| \alpha_1 |1\rangle|^2 \\
&= |\alpha_0 \langle 0| 0\rangle + \alpha_1 \langle 0| 1\rangle|^2 \\
&= |\alpha_0|^2
\end{aligned}
\tag{A.9}
$$

Uma outra maneira de se calcular a probabilidade $p_\psi(x)$ é:

$$\begin{aligned} p_\psi(x) &= |\langle x|\,\psi\rangle|^2 \\ &= \langle x|\,\psi\rangle^*.\,\langle x|\,\psi\rangle \\ &= \langle \psi|\,x\rangle.\,\langle x|\,\psi\rangle \\ &= \langle \psi|\,P_x\,|\psi\rangle \end{aligned} \tag{A.10}$$

com $P_x = |x\rangle\langle x|$. Pode-se reescrever a expressão à direita da equação A.10:

$$\begin{aligned} \langle \psi|\,P_x\,|\psi\rangle &= \langle \psi|\,x\rangle\,\langle x|\,\psi\rangle \\ &= \langle x|\,\psi\rangle\,\langle \psi|\,x\rangle \\ &= \langle x|\,P_\psi\,|x\rangle \end{aligned} \tag{A.11}$$

com $P_\psi = |\psi\rangle\langle\psi|$. Os operadores P_x e P_ψ são chamados de **operadores de projeção**.

A.3 Processamento da Informação

Em um computador clássico, o processamento da informação é realizado por dispositivos chamados de circuitos lógicos, que são agrupamentos de dispositivos mais simples chamados de portas lógicas. Analogamente, a CQ também possui um modelo de circuitos quânticos formado por portas quânticas que realizam operações unitárias sobre os qubits. Uma porta quântica simples aplica uma operação unitária U sobre um qubit no estado $|\psi\rangle$ fazendo-o evoluir para o estado $U|\psi\rangle$. Tais portas são representadas por matrizes unitárias, que preservam a norma do vetor e atendem à seguinte propriedade: $UU^\dagger = U^\dagger U = I$. Desse modo, já que toda matriz unitária, por definição, possui inversa, as operações definidas sobre os qubits são reversíveis.

A.3.1 Portas Quânticas de um Qubit

As portas mais simples são as que operam sobre um único qubit. Por exemplo, a porta X, descrita pela matriz:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{A.12}$$

representa a porta clássica NOT:

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

Outras portas quânticas importantes são apresentadas abaixo e, juntamente com a porta X, formam um conjunto conhecido por matrizes de Pauli:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$
$$Y = \begin{bmatrix} 0 & -i \\ 1 & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Percebe-se que nem todas as portas quânticas possuem correspondentes clássicas. Uma porta de um qubit bastante utilizada na Computação Quântica e que não possui análoga clássica é a porta Hadamard descrita abaixo:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{A.13}$$

Sua ação leva um estado da base a uma superposição de estados, e quando aplicada novamente, leva a superposição ao estado original, como mostrado nas equações A.15 e A.16 a seguir:

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{A.14}$$
$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$H\frac{|0\rangle+|1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)+\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)\right) = |0\rangle \tag{A.15}$$
$$H\frac{|0\rangle-|1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)-\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)\right) = |1\rangle$$

A.3.2 Portas Quânticas de Múltiplos Qubits

As portas descritas na subseção anterior atuam sobre um qubit apenas. No entanto, para realizar qualquer operação quântica são necessárias também portas que atuem sobre múltiplos qubits. A porta mais importante dessa categoria é a porta CNOT ou NOT-controlada. Esta porta define uma operação sobre dois qubits a e b (qubit de controle e qubit alvo, respectivamente) representada através da figura A.3.2, onde a operação $\oplus$ representa o ou-exclusivo.

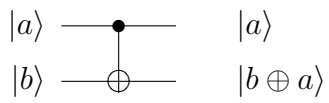


Figura A.1: Circuito que representa a porta CNOT

A matriz que a descreve segue abaixo:

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

A porta CNOT atua da seguinte maneira: se o qubit de controle a está no estado $|0\rangle$, o estado $|b\rangle$ do qubit alvo permanece inalterado; no entanto, se o qubit de controle a está no estado $|1\rangle$ então o estado $|b\rangle$ do qubit alvo é alterado para $|b \oplus a\rangle$. A operação sobre o qubit alvo pode ser expressa como $X^a |b\rangle$ onde X é a porta X descrita na sessão anterior e o sobrescrito a é o valor do estado do bit de controle.

Pode-se generalizar a ação da porta CNOT para uma porta U-controlada, onde U é uma operação unitária sobre um qubit, e sua ação é representada pela figura A.3.2:

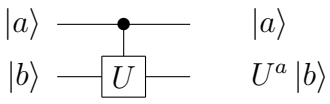


Figura A.2: Circuito que representa a porta U-controlada

É possível controlar a aplicação da porta no qubit alvo quando o qubit de controle está no estado $|0\rangle$. A representação segue na figura A.3.2:

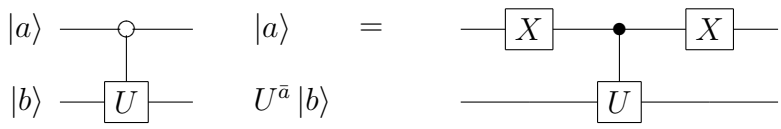


Figura A.3: Circuito que representa a porta U-controlada acionada pelo qubit de controle no estado $|0\rangle$

As portas controladas podem ser expandidas para funcionar com múltiplos qubits de controle. A operação só é aplicada quando todos eles satisfazem os requisitos de entrada. Um exemplo de porta com múltiplos controles é apresentado na figura A.3.2.

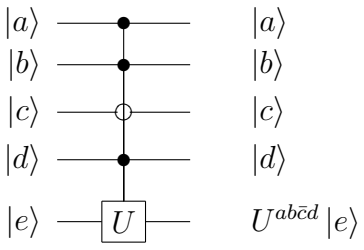


Figura A.4: Circuito que representa a porta U-multicontrolada para 4 qubits

Para esse exemplo, a porta U só será aplicada ao qubit $|e\rangle$ quando $a = |1\rangle$, $b = |1\rangle$, $c = |0\rangle$ e $d = |1\rangle$.

Uma outro exemplo de porta sobre múltiplos qubits que envolve o uso de controles é a porta SWAP. Ela é responsável pela troca de valores entre dois qubits e o circuito que a representa é apresentado na figura A.3.2.

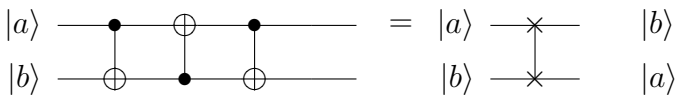


Figura A.5: Circuito que representa a porta SWAP

A ação do circuito é detalhada na equação abaixo:

$$\begin{aligned} |\psi_0\rangle &= |a\rangle\,|b\rangle \\ |\psi_1\rangle &= |a\rangle\,|b \oplus a\rangle \\ |\psi_2\rangle &= |a \oplus (|b \oplus a\rangle)\rangle\,|b \oplus a\rangle = |b\rangle\,|b \oplus a\rangle \\ |\psi_3\rangle &= |b\rangle\,|(b \oplus a) \oplus b\rangle = |b\rangle\,|a\rangle \end{aligned} \tag{A.16}$$

A.4 Computação

Uma operação quântica é unitária e portanto reversível. Logo, um computador quântico precisa de dois registradores para realizar uma computação: um para guardar o estado da entrada e outro para o estado da saída. A computação de uma função f é determinada por uma operação unitária U_f que age sobre os dois registradores preservando a entrada, de acordo com o seguinte protocolo:

$$U_f\,|x\rangle\,|y\rangle = |x\rangle\,|y \oplus f(x)\rangle \tag{A.17}$$

Pode-se observar que, se $y = 0$, então,

$$U_f\,|x\rangle\,|0\rangle = |x\rangle\,|0 \oplus f(x)\rangle = |x\rangle\,|f(x)\rangle \tag{A.18}$$

Para ilustrar uma computação, suponha que é preparado um registrador com m qubits no estado $|\psi\rangle$ de superposição igualmente distribuída e um registrador com um qubit no estado $|0\rangle$, descritos pela equação A.19:

$$|\psi\rangle\,|0\rangle = \frac{1}{2^{m/2}} \sum_{x=0}^{2^m-1} |x\rangle\,|0\rangle \tag{A.19}$$

Aplicando U_f ao estado $|\psi\rangle\,|0\rangle$, como mostra o circuito da figura A.6,

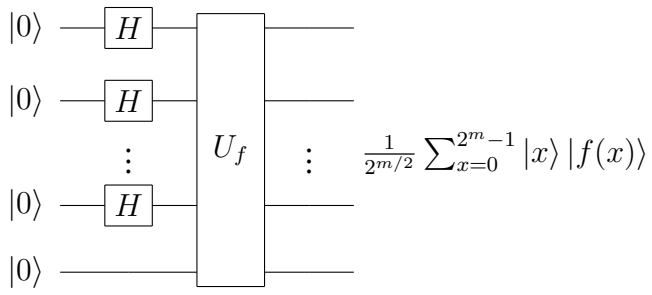


Figura A.6: Circuito que computa o valor de f para todos os valores de x.

obtem-se:

$$\begin{aligned} U_f\,|x\rangle\,|0\rangle &= U_f\left(\frac{1}{2^{m/2}} \sum_{x=0}^{2^m-1} |x\rangle\,|0\rangle\right) \\ &= \frac{1}{2^{m/2}} \sum_{x=0}^{2^m-1} U_f\,|x\rangle\,|0\rangle \\ &= \frac{1}{2^{m/2}} \sum_{x=0}^{2^m-1} |x\rangle\,|f(x)\rangle \end{aligned} \tag{A.20}$$

Ou seja, o circuito realiza a computação de todos os 2^m valores $f(0), f(1), \ldots, f(2^m-1)$ ao mesmo tempo com uma única aplicação da operação unitária U_f que implementa a função f. Essa característica incomum de calcular todos os valores de f ao mesmo tempo é chamada paralelismo quântico. No entanto, a informação só será obtida ao efetuar uma medição, que como visto na seção A.2, é um processo que altera o estado do sistema.

Apêndice B

Circuitos Quânticos

Este apêndice descreve sucintamente o conceito de família de circuitos e os atributos de avaliação de eficiência dos mesmos.

B.1 Família de Circuitos

Circuitos são redes compostas por fios que carregam bits para as portas e estas executam operações elementares nos bits. Os circuitos considerados aqui são acíclicos, ou seja, os bits se movem através do circuito de forma linear, não havendo *loops* nos fios. Um circuito C_n tem n fios e pode ser descrito por um diagrama de circuitos, como, por exemplo, o diagrama da figura B.1 abaixo:

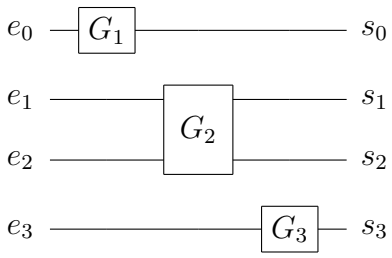


Figura B.1: Exemplo de circuito com $n = 4$.

Os fios são representados pelas linhas horizontais. À esquerda do circuito são escritos os bits de entrada, e a cada instante de tempo t, uma porta G_i é aplicada aos bits. A computação ocorre da esquerda para a direita e, ao final, os bits de saída são lidos à direita do circuito.

Uma **família de circuitos** é um conjunto de circuitos $C_n|n \in Z+$, em que há um circuito para cada entrada de tamanho n. A família é **uniforme** se cada C_n pode ser construído facilmente, isto é, se há uma Máquina de Turing que gera os circuitos em tempo polinomial.

Um conjunto de portas é universal se é possível construir um circuito para realizar qualquer computação utilizando apenas portas daquele conjunto. Para a computação clássica, um exemplo de potas universais é o conjunto $NAND, FANOUT$.

B.2 Complexidade de Circuitos

A complexidade computacional, no modelo de Máquinas de Turing, é especificada em relação ao tempo ou espaço que a máquina utiliza para completar a computação de uma determinada tarefa. Para o modelo de circuitos existem três medidas de complexidade. Uma delas é o **número total de portas** utilizadas no circuito. Outra é a **profundidade** do circuito, isto é, se o circuito é dividido em uma sequência de instantes de tempo t, em que a aplicação de uma única porta requer um único instante t, então a profundidade será o número total de instantes t. Perceba que a quantidade de portas e a profundidade podem ser diferentes, pois pode-se aplicar mais uma porta no mesmo instante t, contanto que elas atuem sobre bits distintos. A terceira medida é a **largura** ou espaço do circuito, calculado pelo número de bits ou fios no circuito. Essas medidas estão apresentadas no exemplo da figura B.2 abaixo:

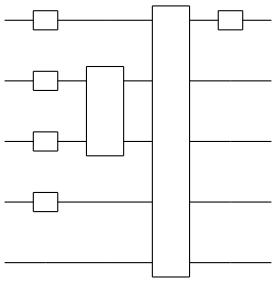


Figura B.2: Circuito com 7 portas, profundidade $= 4$ e largura $= 5$.

Apêndice C

Detalhes da definição 2

Esse apêndice apresenta em detalhes a definição 2 apresentada no capítulo 4, seção 4.2.

Para $w = a_1 a_2 \cdots a_n$ tem-se que $U_w(|q_{init}\rangle) = U_{a_n} \cdots U_{a_2} U_{a_1} |q_{init}\rangle$ é dado por:

$$U_{a_1} |q_{init}\rangle = \sum_{d_1=1}^{|\delta(q_{init},a_1)|} \frac{1}{\sqrt{|\delta(q_{init},a_1)|}} \left|q_{j_{d_1}}\right\rangle$$

, onde $q_{j_{d_1}} \in \delta(q_{init}, a_1)$.

$$\begin{aligned} U_{a_2}(U_{a_1} |q_{init}\rangle) &= \sum_{d_1=1}^{|\delta(q_{init},a_1)|} \frac{1}{\sqrt{|\delta(q_{init},a_1)|}} U_{a_2} \left|q_{j_{d_1}}\right\rangle \\ &= \sum_{d_1=1}^{|\delta(q_{init},a_1)|} \frac{1}{\sqrt{|\delta(q_{init},a_1)|}} \left(\sum_{d_2=1}^{|\delta(q_{j_{d_1}},a_2)|} \frac{1}{\sqrt{|\delta(q_{j_{d_1}},a_2)|}} \left|q_{j_{d_1 d_2}}\right\rangle \right) \end{aligned}$$

, onde $q_{j_{d_1 d_2}} \in \delta(\delta(q_{init}, a_1), a_2)$.

$$\begin{aligned} U_{a_3}(U_{a_2} U_{a_1} |q_{init}\rangle) &= \sum_{d_1=1}^{|\delta(q_{init},a_1)|} \sum_{d_2=1}^{|\delta(q_{j_{d_1}},a_2)|} \frac{1}{\sqrt{|\delta(q_{init},a_1)|}} \frac{1}{\sqrt{|\delta(q_{j_{d_1}},a_2)|}} U_{a_3} \left|q_{j_{d_1 d_2}}\right\rangle \\ &= \sum_{d_1=1}^{|\delta(q_{init},a_1)|} \sum_{d_2=1}^{|\delta(q_{j_{d_1}},a_2)|} \frac{1}{\sqrt{|\delta(q_{init},a_1)|}} \frac{1}{\sqrt{|\delta(q_{j_{d_1}},a_2)|}} \left(\sum_{d_3=1}^{|\delta(q_{j_{d_1 d_2}},a_3)|} \frac{1}{\sqrt{|\delta(q_{j_{d_1 d_2}},a_3)|}} \left|q_{j_{d_1 d_2 d_3}}\right\rangle \right) \end{aligned}$$

, onde $q_{j_{d_1 d_2 d_3}} \in \delta(\delta(\delta(q_{init}, a_1), a_2), a_3)$.

Logo, para U_w, tem-se que:

$$\begin{aligned} U_w(|q_{init}\rangle) &= U_{a_n} \cdots U_{a_2} U_{a_1}(|q_{init}\rangle) \\ &= \sum_{d_1=1}^{|\delta(q_{init},a_1)|} \cdots \sum_{d_n=1}^{|\delta(q_{j_{d_{n-1}}},a_n)|} \frac{1}{\sqrt{|\delta(q_{init},a_1)| \cdots |\delta(q_{j_{d_1 \cdots d_{n-1}}},a_n)}} \left| q_{j_{d_1 \cdots d_n}} \right\rangle \end{aligned}$$