Sistemas interativos de prova clássicos e quânticos

Carlos Henrique Cardonha

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título de mestre
em Ciências

Área: Ciência da Computação
Orientadora: Profa. Dra. Cristina Gomes Fernandes

Durante a elaboração deste trabalho o autor recebeu auxílio financeiro da Fapesp 04/11339-0

São Paulo, setembro de 2006

Sistemas Interativos de Prova Clássicos e Quânticos

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Carlos Henrique Cardonha e aprovada pela Comissão Julgadora.

São Paulo, 18 de setembro de 2006.

Banca Examinadora:

Profa. Dra. Cristina Gomes Fernandes (orientadora) - IME/USP
Prof. Dr. Yoshiharu Kohayakawa - IME/USP
Prof. Dr. Arnaldo Vieira Moura - IC/UNICAMP

Resumo

Baseando-se em discussões relacionadas a simulações de sistemas quânticos, Feynman sugeriu na década de 80 a construção de computadores que pudessem explorar as características quânticas da natureza. A conseqüência disso foi o início do desenvolvimento da teoria de computação quântica, que consiste num modelo de computação possivelmente mais poderoso que o modelo clássico. O algoritmo de Shor para a fatoração de inteiros reforçou as suspeitas a respeito da superioridade desse modelo.

No mesmo período, um novo conjunto de ferramentas foi desenvolvido dentro da teoria de complexidade computacional. Os sistemas interativos de prova foram introduzidos na década de 80 e, com eles, muitos resultados importantes foram obtidos, como o teorema **PCP**.

Recentemente, surgiram alguns novos resultados envolvendo sistemas interativos de prova e o modelo quântico de computação. Esta dissertação apresenta alguns desses resultados com o intuito de evidenciar algumas das potenciais diferenças entre os modelos quântico e clássico de computação.

Abstract

Based on discussions related to quantum system simulations, Feynman suggested in the 80s the construction of computers that would explore the quantum features of nature. As a result, it was developed the theory of quantum computing, which consists in a possibly more powerful model of computation than the classical model. Shor's algorithm for integer number factorization enhanced the suspicion about the superiority of this model.

In the same period, a new set of tools was developed within the computational complexity theory. The interactive proof systems were introduced in the 80s, and with them many important results were obtained, as the **PCP** theorem.

Recently, some new results involving interactive proof systems and quantum computing were presented. This dissertation shows some of these results to make comparisons between the quantum and the classic model of computation.

Sumário

Capítulo 1

Introdução

Em 1900, em uma palestra no Congresso Internacional de Matemáticos em Paris, Hilbert postulou 23 problemas matemáticos, que tratavam de temas diversos em matemática e áreas afins. O décimo problema na lista de Hilbert (*determination of the solvability of a diophantine equation*) pergunta se é possível determinar se uma equação diofantina arbitrária tem ou não solução por meio de um "processo finito":

> *Given a diophantine equation with any number of unknown quantities and with rational numerical coefficients: to devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.*

Esse problema pode ser postulado em uma linguagem mais atual como o seguinte: existe um algoritmo que, dada uma equação diofantina, determina se esta tem ou não solução? Matiyasevich [Mat70] mostrou que a resposta é negativa.

Note que a questão postulada por Hilbert precede de décadas a invenção de computadores. Foi apenas nos anos 30 que tais questões foram formuladas e tratadas dentro do que ficou depois conhecido como *teoria da computabilidade*. Esta é a parte da teoria da computação especializada em lidar com esse tipo de questão.

Foi nos anos 30, após um trabalho de Gödel em lógica, que a idéia de algoritmo começou a ser formalizada. Gödel [Göd31] introduziu o conceito de *função primitiva recursiva* como uma formalização dessa idéia. Church [Chu33, Chu36] introduziu o λ-*cálculo* e Kleene [Kle36] definiu o conceito de *funções recursivas parciais* e mostrou a equivalência entre esse e o λ-cálculo. Turing [Tur36, Tur37] por sua vez propôs a sua formalização da idéia de algoritmo: as chamadas *máquinas de Turing*. Vale mencionar que outro modelo de poder equivalente ao

das máquinas de Turing foi independentemente proposto por Post [Pos36], um professor de colegial de Nova Iorque. Cada uma dessas propostas diferentes do conceito de algoritmo é chamada de *modelo de computação*.

Foi Kleene [Kle52] quem chamou de *tese de Church* a afirmação de que todo modelo de computação "razoável" é equivalente ao da máquina de Turing. A afirmação é propositalmente vaga, pois visa capturar mesmo modelos que ainda venham a ser propostos, e cuja natureza não podemos prever. Por "razoável" entende-se um modelo que seja realista, no sentido de poder ser construído na prática (mesmo que de maneira aproximada).

A teoria da computabilidade no fundo diferencia os problemas *decidíveis* (para os quais existe um algoritmo) dos *indecidíveis* (para os quais não existe um algoritmo). O surgimento dos computadores nas décadas de 30 e 40 aos poucos evidenciou uma diferença entre os problemas decidíveis: muitos parecem ser bem mais difíceis que outros, no sentido de que se conhece apenas algoritmos extremamente lentos para eles. Com isso, surgiu a necessidade de refinar a teoria de computabilidade para tentar explicar essas diferenças. Foi apenas nos anos 60 que a *teoria de complexidade*, que trata de tais questões, tomou corpo, com a formalização da idéia de "algoritmo eficiente", independentemente introduzida por Cobham [Cob65] e Edmonds [Edm65], e a proposta de reduções eficientes entre problemas [Kar72].

Foi nessa época que surgiram as definições das classes de complexidade **P** e **NP** e do conceito de **NP**-*completude*, que captura de certa maneira a dificuldade de se conseguir algoritmos eficientes para certos problemas. Grosseiramente, um problema em **NP** é dito **NP**-*completo* se qualquer outro problema da classe **NP** pode ser reduzido eficientemente a ele. A mais famosa questão na área de teoria da computação é se **P** é ou não igual a **NP**. Se for mostrado que algum problema **NP**-completo está em **P**, então tal questão é resolvida e fica provado que $\mathbf{P} = \mathbf{NP}$.

Um marco na teoria de complexidade é o *teorema de Cook* [Coo71, Lev73], que prova a existência de problemas **NP**-completos. Cook mostrou que o problema conhecido como SAT, de decidir se uma fórmula booleana em forma normal conjuntiva é ou não satisfatível, é **NP**-completo. Após o teorema de Cook e o trabalho de Karp [Kar72], que mostrou que vários outros problemas conhecidos de otimização combinatória eram **NP**-completos, essa teoria se desenvolveu amplamente, tendo estabelecido a dificuldade computacional de problemas das mais diversas áreas [GJ79].

Um problema muito famoso cuja complexidade continua em aberto, mesmo após várias décadas de esforço da comunidade no sentido de resolvê-lo, é o problema da *fatoração de inteiros*: dado um inteiro, determinar a sua fatoração em números primos. Recentemente, o seu parente próximo, o problema de decidir

se um número inteiro é primo ou não, chamado de *problema da primalidade*, teve sua complexidade totalmente definida, com o algoritmo AKS, de Agrawal, Kayal e Saxena [AKS02a, AKS02b]. Esse algoritmo mostra que o problema da primalidade está na classe **P**, resolvendo com isso uma questão em aberto há anos. Não se sabe até hoje, no entanto, se há um algoritmo eficiente para resolver o problema da fatoração de inteiros!

Na verdade, a dificuldade computacional do problema da fatoração de inteiros tem sido usada de maneira crucial em alguns sistemas criptográficos bem-conhecidos. Se for descoberto um algoritmo eficiente para resolver o problema da fatoração, vários sistemas criptográficos importantes seriam quebrados, incluindo o famoso sistema RSA de chave pública criado por Rivest, Shamir e Adleman [RSA78].

Nesse trabalho, discutimos um modelo de computação mais recente, o modelo quântico, que vem levantando questões intrigantes dentro da teoria de complexidade e que pode ter impactos práticos dramáticos na área de criptologia. O modelo quântico de computação não infringe a validade da tese de Church, porém questiona a validade de uma versão mais moderna dessa, a chamada *tese de Church estendida*, que diz que todo modelo de computação razoável pode ser simulado *eficientemente* por uma máquina de Turing.

Pode-se dizer que a teoria de computação quântica iniciou-se nos anos 80, quando Feynman [Fey82] observou que um sistema quântico de partículas, ao contrário de um sistema clássico, parece não poder ser simulado eficientemente em um computador clássico e sugeriu um computador que explorasse efeitos da física quântica para contornar o problema. Desde então, até 1994, a teoria de computação quântica desenvolveu-se discretamente, com várias contribuições de Deutsch [Deu85, Deu89], Bernstein e Vazirani [BV97], entre outros, que colaboraram fundamentalmente para a formalização de um modelo computacional quântico.

Foi apenas em 1994 que a teoria recebeu um forte impulso e uma enorme divulgação. Isso deveu-se ao algoritmo de Shor [Sho94, Sho97], um algoritmo quântico eficiente para o problema da fatoração de inteiros, considerado o primeiro algoritmo quântico combinando relevância prática e eficiência. O algoritmo de Shor é uma evidência de que o modelo computacional quântico proposto pode superar de fato o modelo clássico, derivado das máquinas de Turing. O resultado de Shor impulsionou tanto a pesquisa prática, objetivando a construção de um computador segundo o modelo quântico, quanto a busca por algoritmos criptográficos alternativos e algoritmos quânticos eficientes para outros problemas difíceis. Essas e várias outras questões, relacionadas tanto com a viabilidade do modelo quântico quanto com as suas limitações, têm sido objeto de intensa pesquisa científica.

Do ponto de vista prático, busca-se descobrir se é ou não viável construir um computador segundo o modelo quântico que seja capaz de manipular números suficientemente grandes. Tal viabilidade esbarra em uma série de questões técnicas e barreiras físicas e tecnológicas. Já se tem notícia de computadores construídos segundo o modelo quântico, mas todos ainda de pequeno porte. Em 2001, por exemplo, foi construído um computador quântico com 7 *qubits* (o correspondente aos bits dos computadores tradicionais). Nesse computador, foi implementado o algoritmo de Shor que, nele, fatorou o número 15. Uma parte dos cientistas da computação acredita que a construção de computadores quânticos de maior porte será possível, enquanto outra parte não acredita nisso.

Do ponto de vista de teoria de complexidade, busca-se estabelecer a relação entre as classes de complexidade derivadas do modelo quântico e as classes de complexidade tradicionais. Também busca-se, claro, estabelecer a complexidade no modelo quântico de problemas bem-conhecidos, ou seja, busca-se algoritmos quânticos eficientes para outros problemas relevantes.

Várias ferramentas têm sido utilizadas para o estabelecimento da relação entre as classes de complexidade clássicas e as classes de complexidade quânticas. Algumas delas são os chamados sistemas interativos de prova.

Os sistemas interativos de prova foram introduzidos em meados da década de 80 [Bab85, GMR85], tendo como motivação original aplicações em criptografia. Porém, o uso desses sistemas também levou a uma série de resultados importantes em complexidade computacional. Em especial, os sistemas interativos levaram ao desenvolvimento das provas verificáveis probabilisticamente [ALM$^+$92]. Esses resultados culminaram em uma caracterização alternativa da classe **NP**. Uma das conseqüências de tal caracterização foi a demonstração da inaproximabilidade de uma série de problemas de otimização combinatória bem conhecidos.

Recentemente, foram apresentadas versões dos sistemas interativos e das provas verificáveis probabilisticamente para o modelo quântico [Wat03, MW04, Raz05], e alguns resultados envolvendo tais sistemas já foram descobertos. A comparação dos resultados conhecidos no momento envolvendo sistemas interativos clássicos e sistemas interativos quânticos compõe mais uma evidência de que o modelo quântico pode superar o modelo clássico de computação.

O objetivo desse trabalho é apresentar de maneira didática as definições do modelo clássico e do modelo quântico de computação, bem como algumas das principais relações conhecidas envolvendo classes de complexidade dos dois modelos por meio dos sistemas interativos de prova.

Organização do texto

Esse texto está dividido em duas partes. A primeira concentra-se em definições e resultados do modelo clássico de computação, e a segunda nas definições e resultados relacionados que aparecem no modelo quântico de computação.

A parte clássica começa no capítulo 2, com a descrição de elementos básicos de complexidade computacional, apresentando os conceitos de máquinas de Turing, de consumo de espaço e tempo e as definições de algumas classes de complexidade que serão estudadas nesse trabalho: além de **P** e **NP**, as classes **coNP**, **PSPACE**, **BPP** e outras são introduzidas. No capítulo 3, apresentamos a hierarquia polinomial e sua relação com as classes **PSPACE** e **BPP**. Introduzimos no capítulo 4 os sistemas interativos de prova e apresentamos alguns dos resultados mais importantes envolvendo tais sistemas: **coNP** $\subseteq$ **IP** e **IP** = **PSPACE**, onde **IP** é a classe composta por linguagens decididas por sistemas interativos de prova que utilizam um número polinomial de trocas de mensagens.

A segunda parte começa no capítulo 5, com uma apresentação do modelo quântico de computação. Inicialmente são descritas algumas características especiais da computação quântica, e em seguida o modelo quântico de computação baseado em circuitos é apresentado. Também mostramos um exemplo simples de algoritmo quântico e as definições de algumas classes de complexidade envolvendo o modelo quântico: **EQP** e **BQP**.

No capítulo 6 são introduzidos os sistemas interativos quânticos de prova e um resultado importante é apresentado: **PSPACE** $\subseteq$ **QIP**(3), onde **QIP**(3) é a classe composta por linguagens decididas por sistemas interativos quânticos que utilizam três trocas de mensagens.

No capítulo 7 algumas propriedades dos **SIQP** decorrentes das propriedades do modelo quântico são mostradas. Em especial, apresentamos o seguinte resultado: **QIP** $\subseteq$ **QIP**(3), onde **QIP** é a classe composta por linguagens decididas por sistemas interativos quânticos que utilizam um número polinomial de trocas de mensagens.

O capítulo 8 contém alguns comentários finais.

Consideramos como nossas contribuições principais as apresentações dos resultados **IP** = **PSPACE**, **PSPACE** $\subseteq$ **QIP**(3) e **QIP** $\subseteq$ **QIP**(3). A demonstração aqui apresentada de que **IP** = **PSPACE** inspirou-se fortemente no livro de Kohayakawa e Soares [KS95], porém optamos por apresentar um dos algoritmos envolvidos na prova de maneira diferente, que nos pareceu mais clara. Já os dois outros resultados aparecem apenas nos artigos originais. A nossa apresentação deles é bem mais detalhada e, esperamos, mais acessível ao leitor.

Capítulo 2

Preliminares

Nesse capítulo, apresentamos os conceitos e os resultados da teoria clássica de complexidade computacional que serão úteis nos estudos da hierarquia polinomial e dos sistemas interativos de provas.

Inicialmente, apresentamos três variantes de máquinas de Turing. Em seguida, apresentamos algumas classes de complexidade, bem como uma série de outros conceitos importantes da área que serão utilizados nesse texto.

2.1 Máquinas de Turing

No estudo da teoria clássica de complexidade, tradicionalmente utiliza-se como modelo de computação a máquina de Turing (MT), que pode ser vista como um computador bastante rudimentar com memória infinita.

Vamos apresentar três variantes desse modelo: a máquina de Turing determinística, a não-determinística e a probabilística. Tais máquinas serão utilizadas mais adiante, quando definirmos as classes de complexidade em que estamos interessados do modelo clássico.

Máquina de Turing determinística

Uma *máquina de Turing determinística* (MTD) é composta por uma central de controle, uma cabeça de leitura e uma fita dividida em células. Essa fita tem um final à esquerda e contém infinitas células à direita.

Cada célula da fita armazena um símbolo pertencente a um conjunto finito Σ'. O conjunto Σ' é chamado de *alfabeto* da máquina e contém, entre outros, dois símbolos especiais: o símbolo $\sqcup$, chamado de *branco*, e o símbolo $\triangleright$, que fica armazenado durante toda a computação na célula mais à esquerda da

fita. Vamos denotar por Σ o conjunto formado pelos outros símbolos de Σ'. Ou seja, $\Sigma = \Sigma' \setminus \{\triangleright, \sqcup\}$.

A cabeça de leitura da máquina é um apontador móvel que em cada passo está sobre uma determinada célula da fita. O conteúdo dessa célula pode ser lido e alterado pela máquina nesse passo.

Em cada passo, a central de controle da máquina está em um dos estados de um conjunto finito Q. A máquina começa a computação em um estado particular de Q chamado *estado inicial*, que é denotado por s. Existe ainda um conjunto especial de estados $H \subseteq Q$, chamados de *estados finais*, que quando atingidos resultam na parada da máquina.

Uma MTD funciona da seguinte maneira. Ela recebe como entrada uma cadeia de caracteres em Σ^*. Inicialmente a cabeça de leitura aponta para a célula mais à esquerda da fita (a que contém o símbolo $\triangleright$). A partir da célula seguinte está armazenada a entrada, seguida por brancos. A máquina efetua uma seqüência de passos até terminar a execução. Se q é o estado corrente da máquina e q está em H, então ela pára. Do contrário, ela realiza três ações, determinadas pelo par (q, σ), onde σ é o símbolo de Σ' contido na célula que está sob a cabeça de leitura.

A primeira ação consiste na alteração do estado da máquina de q para um estado q' em Q. A segunda ação é a escrita de um símbolo σ' de Σ' na célula que está sob a cabeça de leitura. A terceira ação consiste no deslocamento da cabeça de leitura, que pode se mover uma posição para a esquerda, uma posição para a direita ou pode continuar apontando para a mesma célula.

A cabeça de leitura da fita sempre desloca-se para a direita quando atinge a célula mais à esquerda e nunca altera o conteúdo dessa posição da fita. Por conveniência, assume-se que a máquina não pode escrever o símbolo $\triangleright$ em qualquer posição da fita que não seja a célula mais à esquerda. Além disso, quando um estado final é atingido, a transição definida para esses casos mantém o estado, o conteúdo da célula e a posição da cabeça de leitura. Por isso dizemos que nesse caso a máquina pára.

A cadeia de caracteres escrita na fita quando a máquina termina a execução, ignorando-se o símbolo $\triangleright$ e os brancos à direita, é a saída da máquina para a entrada em questão.

Formalmente, define-se uma máquina de Turing determinística como uma quíntupla $M = (Q, \Sigma', \delta, s, H)$, onde Q é o conjunto de estados, Σ' é o alfabeto de símbolos, δ é uma função de $(Q \setminus H) \times \Sigma'$ em $Q \times \Sigma' \times \{\leftarrow, \downarrow, \rightarrow\}$, $s \in Q$ é o estado inicial e $H \subseteq Q$ é o conjunto de estados finais. O conjunto $\{\leftarrow, \downarrow, \rightarrow\}$ descreve os possíveis movimentos da cabeça de leitura da máquina.

A função δ, chamada usualmente de *função de transição*, descreve cada um

dos possíveis passos da máquina e satisfaz as restrições mencionadas acima. Suponha que a máquina esteja num estado q em $Q \setminus H$ e que sob a cabeça de leitura esteja o símbolo σ. Se $\delta(q, \sigma) = (q', \sigma', d)$, o próximo estado será q', o símbolo σ' será escrito no lugar de σ e a cabeça de leitura de M moverá de acordo com d.

A *configuração atual* de M em um passo é uma tripla $(w_1, q, w_2) \in (\Sigma')^* \times Q \times (\Sigma')^*$, onde w_1 é a palavra que aparece na fita à esquerda da cabeça de leitura, q é o estado atual e w_2 é a palavra à direita da cabeça de leitura ignorando-se brancos à direita. A cabeça de leitura aponta para a posição que contém o último símbolo de w_1. A configuração inicial é a tripla $(\triangleright, s, x)$, onde x é a entrada para a máquina.

Dizemos que uma configuração (w_1, q, w_2) *produz em k passos* uma configuração (w_1', q', w_2'), o que é denotado por $(w_1, q, w_2) \vdash_M^k (w_1', q', w_2')$, se a máquina sai da primeira configuração e vai para a segunda em exatos k passos. Se q' é um estado final, sempre que nos referirmos ao número k em $(w_1, q, w_2) \vdash_M^k (w_1', q', w_2')$ vamos nos referir ao menor inteiro k para o qual tal relação é verdadeira. Dessa forma, limitamos também o número de passos executados por uma MTD (a contagem dos passos pára quando um estado final é atingido).

Máquina de Turing não-determinística

A *máquina de Turing não-determinística* (MTND) é uma generalização da máquina de Turing determinística. Essa máquina tem um papel fundamental na teoria de complexidade pois está na base da definição da classe **NP**, conforme será mostrado posteriormente.

A maior parte das definições e idéias envolvidas na descrição das MTDs também se aplica às MTNDs. A diferença entre a MTND e a MTD está na função de transição e na maneira como as transições são feitas a cada passo. Nas máquinas determinísticas, existe uma única transição possível a partir de uma dupla (q, σ), onde q é um estado não-final e σ é um símbolo em Σ'. Já nas máquinas não-determinísticas, pode existir mais de uma transição válida a partir de uma dupla (q, σ). Em cada passo da MTND, uma transição válida é escolhida arbitrariamente para ser realizada.

O número de transições válidas a partir de uma dupla (q, σ) é limitado superiormente por $3 \times |\Sigma'| \times |Q|$. Logo, o número de configurações que podem ser produzidas a partir de cada configuração também é limitado superiormente por $3 \times |\Sigma'| \times |Q|$. A transição numa MTND é portanto uma *relação* e não necessariamente uma função.

Formalmente, uma máquina de Turing não-determinística é uma quíntupla

$M = (Q, \Sigma', \Delta, s, H)$, onde Q é o conjunto de estados, Σ' é o alfabeto de símbolos, Δ é uma relação de $(Q \setminus H) \times \Sigma'$ em $Q \times \Sigma' \times \{\leftarrow, \downarrow, \rightarrow\}$, $s \in Q$ é o estado inicial e $H \subseteq Q$ é o conjunto de estados finais.

É evidente que as MTDs são casos particulares de MTNDs em que Δ é uma função, já que toda função é uma relação.

Uma transição (q', σ', d) em $\Delta(q, \sigma)$, onde $q \in Q$ e $\sigma \in \Sigma'$, é interpretada como antes. Assim, q' será o próximo estado da máquina, σ' deve ser escrito no lugar de σ e d indicará o deslocamento da cabeça de leitura. Tal transição será válida somente se $(q', \sigma', d) \in \Delta(q, \sigma)$. A definição de configuração é igual à que usamos na definição da máquina de Turing determinística. Para MTNDs, dizemos que uma configuração (w_1, q, w_2) produz a configuração (w_1', q', w_2') em k passos se, estando a máquina na primeira configuração, após k passos ela pode estar na segunda a partir de uma seqüência de transições válidas.

A existência de várias transições possíveis para cada par (q, σ) possibilita a existência de computações distintas para uma mesma entrada por uma MTND. Conseqüentemente, elas podem produzir saídas diferentes para uma mesma entrada. Comentaremos mais sobre isso quando falarmos das linguagens decididas por MTs.

Máquina de Turing probabilística

Uma *máquina de Turing probabilística* (MTP) é basicamente uma MTND $M = (Q, \Sigma', \Delta, s, H)$ em que o não-determinismo é substituído pelas escolhas probabilísticas.

A cada passo, em vez de uma transição ser escolhida arbitrariamente dentre todas as transições aplicáveis, a escolha é feita com probabilidade uniforme. Assim, pode-se falar na probabilidade da MTP produzir, numa computação para uma certa entrada, uma saída específica. Todas as definições dadas para as MTNDs aplicam-se de maneira natural a uma MTP. Observe que uma MTD é uma MTP em que, a cada passo, há apenas uma transição aplicável.

Uma MTP corresponde à formalização do conceito de um computador acoplado a um gerador de números aleatórios.

Podemos assumir sem perda de generalidade que o número de transições possíveis em cada passo de uma MTP é igual a dois, e que as transições são escolhidas com probabilidade $1/2$. Para fazer essas escolhas, a máquina utiliza uma seqüência de bits aleatórios cujo tamanho é igual ao número de passos da máquina.

2.2 Teoria clássica de complexidade

Nessa seção, apresentamos algumas definições e resultados do modelo clássico de computação importantes para o que apresentaremos a seguir, como a hierarquia polinomial e os sistemas interativos de prova.

2.2.1 Definição dos parâmetros

Apresentamos agora os principais parâmetros em função dos quais as classes de complexidade são definidas, bem como a relação entre linguagens e máquinas de Turing.

Tempo consumido pelas máquinas de Turing

Na descrição das MTDs, foi definido que $(w_1, q, w_2) \vdash_M^k (w_1', q', w_2')$ se a máquina M sai da primeira configuração e vai para a segunda em exatos k passos. Se (w_1, q, w_2) é a configuração inicial de M e q' é um de seus estados finais, então dizemos que k é o tempo consumido por M para a entrada w_2.

Na descrição da MTND, vimos que $(w_1, q, w_2) \vdash_M^k (w_1', q', w_2')$ se existe uma computação válida da máquina M que sai da primeira configuração e vai para a segunda em exatos k passos. Se (w_1, q, w_2) é a configuração inicial de M e q' é um de seus estados finais, então dizemos que existe uma computação válida de M para a entrada w_2 que consome tempo k. O maior valor de k que leva uma MTND M com entrada w de seu estado inicial para um estado final, em qualquer computação válida, é considerado o tempo consumido por M para essa entrada.

Por fim, se M é uma MTP, valem as mesmas definições usadas para as MTNDs. Vale destacar que no caso das MTPs eventualmente aplica-se o conceito de tempo *esperado* de computação, que formalmente é a esperança do tempo consumido por uma computação de M para uma dada entrada x.

Dada uma MT M, se existe uma função $p : \mathbb{N} \to \mathbb{N}$ tal que, para qualquer entrada x, o tempo consumido por M é limitado superiormente por $p(|x|)$, onde $|x|$ denota o comprimento da palavra x, então dizemos que M consome tempo $O(p(n))$. Se $p(n)$ é um polinômio em n, dizemos que M é *polinomialmente limitada*.

Espaço consumido pelas máquinas de Turing

Em complexidade computacional, eventualmente temos interesse no consumo de espaço por máquinas de Turing. Enquanto o tempo é medido em função

do número de passos utilizados pelas MTs, o espaço é medido em função das células de sua fita acessadas durante a computação.

Mais precisamente, dizemos que o *espaço utilizado* por uma MT é a maior quantidade de células distintas usadas pela máquina para realizar a computação de uma determinada entrada. No caso das MTDs, é o número de células usadas na única computação possível. Já para MTNDs e MTPs, é o maior número de células utilizadas em uma computação válida.

Como em toda MT a cabeça de leitura só pode se deslocar de uma célula a cada passo, claramente o espaço utilizado em uma computação é limitado superiormente pelo número de passos executados pela máquina.

Dada uma MT M, se existe uma função $p : \mathbb{N} \to \mathbb{N}$ tal que, para qualquer entrada x, o espaço consumido por M é limitado superiormente por $p(|x|)$, então dizemos que M consome espaço $O(p(n))$. Se $p(n)$ é um polinômio em n, dizemos que M consome *espaço polinomial*.

Linguagens e máquinas de Turing

Ao definir as máquinas de Turing, utilizamos como parâmetro um alfabeto $\Sigma' = \Sigma \cup \{\triangleright, \sqcup\}$, onde Σ é o conjunto de caracteres que contém os símbolos que podem compor uma entrada para essa máquina. Um conjunto de palavras cujos caracteres pertencem a um alfabeto Σ é uma *linguagem* em Σ^*. Um conjunto de linguagens forma uma *classe de complexidade*.

Geralmente estamos interessados em verificar se uma determinada palavra pertence ou não a uma particular linguagem L. Máquinas de Turing que devolvem respostas binárias podem ser usadas para efetuar essa tarefa. Uma das respostas indica que a palavra fornecida como entrada pertence à linguagem L (*aceitação*) e a outra que a palavra não pertence (*rejeição*).

Daqui para a frente, utilizaremos a seguinte definição: um problema que consiste em decidir se uma palavra pertence ou não a uma linguagem (ou seja, um problema que tem um conjunto binário de respostas) será um *problema de decisão*.

Mostraremos agora as relações entre decisão de linguagens e as máquinas de Turing determinísticas, não-determinísticas e probabilísticas.

Se M é uma MTD que devolve como resposta 0 ou 1, que indicam aceitação e rejeição da entrada respectivamente, o conjunto das palavras aceitas por M formam uma linguagem L. Dizemos nesse caso que M *decide* L. Os símbolos 0 e 1 devem pertencer a Σ e devem aparecer sozinhos na fita da máquina na segunda célula da esquerda para a direita após sua parada.

No caso das MTNDs, já vimos que podem existir várias computações e

várias saídas possíveis para uma máquina e uma entrada. Dizemos que uma MTND M decide uma linguagem L se toda palavra $x \in L$ é aceita por M em alguma de suas computações, e se toda palavra $x \notin L$ é rejeitada por M em todas as suas computações.

Por fim, a decisão de linguagens por MTPs tem um aspecto um pouco diferente das demais máquinas de Turing. Assim como as MTNDs, as MTPs também podem produzir respostas distintas para uma mesma entrada. Porém, nas MTPs, cada computação válida (e conseqüentemente cada possível resposta) ocorre com uma determinada probabilidade. A decisão de uma linguagem por uma MTP envolve as probabilidades de obtenção das respostas.

Em alguns casos estamos interessados em fixar uma probabilidade máxima para as rejeições incorretas, em outros para as aceitações incorretas e em outros para as duas simultaneamente. A decisão de linguagens em MTPs, portanto, não possui uma definição única como no caso das MTDs e das MTNDs. Logo, sempre que esse conceito for utilizado, a interpretação desejada será apresentada.

2.2.2 Algumas classes de complexidade

Nessa seção, definimos algumas classes de complexidade utilizando as máquinas de Turing apresentadas. Conceitos importantes como os de redução, completude e certificados sucintos também são apresentados.

As classes P e PSPACE

A partir do conceito de máquinas de Turing polinomialmente limitadas, vamos definir as linguagens e as relações *polinomialmente decidíveis*. Dizemos que uma linguagem é polinomialmente decidível se existe uma MTD polinomialmente limitada que a decide. Analogamente, dada uma relação $R \subseteq (\Sigma^*)^i$ para um inteiro $i \geq 1$, dizemos que R é polinomialmente decidível se existe uma MTD polinomialmente limitada que decide a pertinência nessa relação.

A classe **P** (*polynomial-time*) é o conjunto de todas as linguagens polinomialmente decidíveis. Dizemos que uma linguagem pertence à classe **PSPACE** (*polynomial-space*) se ela é decidida por uma máquina de Turing (determinística, não-determinística ou probabilística) que utiliza espaço polinomial no tamanho da entrada.

É fácil ver que $\mathbf{P} \subseteq \mathbf{PSPACE}$: como já observamos, toda MTD que consome tempo polinomial certamente consome espaço polinomial, já que, a cada passo, a cabeça de leitura da máquina só pode se mover de no máximo uma célula à direita. Por outro lado, é fácil imaginar uma MTD que consome tempo

super-polinomial mas espaço polinomial. Assim, é concebível (e até de se esperar) que existam linguagens em **PSPACE** que não estejam em **P**. Surpreendentemente, não se sabe até hoje se este é o caso ou não. Ou seja, não se conhece uma prova de que $\mathbf{P} \neq \mathbf{PSPACE}$.

Reduções e completude

Nos estudos de classes de complexidade, dois conceitos possuem grande importância: *redução* e *completude*.

O conceito de redução que apresentaremos foi introduzido por Karp [Kar72]. Por isso, eventualmente tais reduções também são denominadas reduções de Karp. Porém, vale destacar que as reduções são muito comuns na matemática, e que tal conceito já vinha sendo utilizado há muito tempo (antes da formalização da teoria de complexidade computacional).

Dizemos que uma linguagem $L_1 \subseteq \Sigma^*$ pode ser reduzida (admite uma *redução de Karp*) a uma linguagem $L_2 \subseteq \Sigma^*$ se existe uma função $R : \Sigma^* \to \Sigma^*$ computável por uma MTD polinomialmente limitada tal que, para todo $x \in \Sigma^*$, vale o seguinte: Se $x \in L_1$, então $R(x) \in L_2$, e se $x \notin L_1$, então $R(x) \notin L_2$.

A partir do conceito de redução, apresentamos o conceito de problemas difíceis para uma classe. Seja $\mathbf{C}$ uma classe de complexidade e seja L uma linguagem. Dizemos que L é $\mathbf{C}$-*difícil* se qualquer linguagem $L' \in \mathbf{C}$ pode ser reduzida a L. Por fim, se L é uma linguagem $\mathbf{C}$-difícil e se $L \in \mathbf{C}$, então dizemos que L é $\mathbf{C}$-*completa*. Similarmente, dizemos que o problema de decisão de L é $\mathbf{C}$-completo ou $\mathbf{C}$-difícil.

Determinar problemas completos para classes de complexidade é importante, pois mostra de maneira clara a dificuldade inerente à classe. Um exemplo de problema **PSPACE**-completo é o QBF (*Quantified Boolean Formula*). Uma instância do QBF é uma fórmula booleana quantificada

$$(Q_1 x_1)(Q_2 x_2) \ldots (Q_n x_n) B(x_1, x_2, \ldots, x_n)$$

onde cada Q_i representa ou $\exists$ ou $\forall$, e $B(x_1, \ldots, x_n)$ é uma fórmula booleana na forma normal conjuntiva. Dada uma instância desse problema, queremos saber se a afirmação descrita é verdadeira ou não.

Uma demonstração do fato de que o problema QBF é **PSPACE**-completo pode ser vista no livro de Papadimitriou [Pap94]. Mais à frente, veremos a utilização desse problema em demonstrações envolvendo sistemas interativos de prova.

As classes NP e coNP

Podemos definir **NP** (*nondeterministic polynomial-time*) em função de MTNDs. Dizemos que uma linguagem pertence à classe **NP** se ela pode ser decidida por uma máquina de Turing não-determinística polinomialmente limitada.

A classe das linguagens cujo complemento pertence a **NP** é denominada **coNP**, onde o *complemento* de uma linguagem L sob um alfabeto Σ é a linguagem $\Sigma^* \setminus L$, que também denotamos por **co**L. De maneira geral, o *complemento* de uma classe de complexidade arbitrária **C** é denotado por **coC** e definido como o conjunto das linguagens cujo complemento está em **C**.

As classes **NP** e **coNP** contêm vários problemas para os quais não se conhece algoritmo polinomial. Essas classes são importantes porque contêm uma grande quantidade de problemas de interesse prático.

Claramente, toda linguagem que está em **P** também está em **NP** e em **coNP**, visto que uma MTD é uma MTND. A questão mais importante e conhecida é se **P** é igual a **NP**. Há vários indícios que sugerem a desigualdade entre as duas classes. Se $\mathbf{P} = \mathbf{NP}$, saberemos que muitos problemas para os quais hoje não se conhecem algoritmos exatos razoáveis têm uma solução polinomial. Porém, são poucos os que acreditam nessa possibilidade.

Certificados sucintos

Vimos acima a definição da classe **NP** em função de máquinas de Turing não-determinísticas. Também podemos caracterizar tal classe utilizando os conceitos que apresentamos abaixo. Uma demonstração da equivalência entre as duas definições pode ser vista em [Pap94].

Inicialmente, vamos definir as relações polinomialmente balanceadas. Seja $R \subseteq (\Sigma^*)^{i+1}$ para algum inteiro $i \geq 0$ uma relação. Se para todo $(x, y_1, \ldots, y_i) \in R$ é verdade que $|y_j| \leq |x|^k$ para algum inteiro $k \geq 1$, para todo j, $1 \leq j \leq i$, então dizemos que R é uma *relação polinomialmente balanceada*. Se a pertinência em R pode ser verificada por uma máquina de Turing determinística polinomialmente limitada, então dizemos que R é uma *relação polinomialmente decidível*.

A classe **NP** pode ser definida em função de relações polinomialmente balanceadas e polinomialmente decidíveis ou, equivalentemente, de certificados sucintos. Se $L \subseteq \Sigma^*$ é uma linguagem, dizemos que $L \in \mathbf{NP}$ se e somente se existe uma relação $R \subseteq \Sigma^* \times \Sigma^*$ polinomialmente decidível e polinomialmente balanceada tal que $L = \{x : (x, y) \in R \text{ para algum } y\}$. Uma palavra y tal que $(x, y) \in R$ é dita um *certificado sucinto* da pertinência de x em L. Nos casos em que R é uma relação $(i+1)$-ária, as últimas i coordenadas compõem

o certificado de pertinência para a primeira coordenada.

Mais adiante, veremos que os certificados sucintos são uma ferramenta importante no estudo da hierarquia polinomial.

A classe BPP

A classe **BPP** (*bounded-error probabilistic polynomial-time*) contém as linguagens para as quais existem MTPs polinomialmente limitadas que devolvem respostas erradas (tanto rejeições como aceitações) com probabilidade estritamente menor que 0,5. Na verdade, qualquer delimitação polinomial dessa probabilidade no intervalo $(0; 0{,}5)$ resultaria na mesma classe. Porém, Papadimitriou [Pap94] mostra que delimitações maiores ou iguais a 0,5 podem levar a uma classe diferente.

A definição da classe **BPP** é simétrica, pois rejeição e aceitação são feitas corretamente na maioria absoluta das computações feitas pelas MTPs envolvidas. Quando existe tal simetria, a classe é igual ao seu complemento. Logo, $\mathbf{BPP} = \mathbf{coBPP}$.

Destacamos que não se sabe se a classe **BPP** está contida na classe **NP**. Também não se sabe se **BPP** possui problemas completos. Porém, é importante destacar o papel "prático" de **BPP**. Informalmente, podemos dizer que os problemas para os quais existem algoritmos de eficiência satisfatória estão em **BPP** (obter eficientemente uma solução correta com altíssima probabilidade é algo bastante satisfatório). Isso mostra a importância do estudo da relação dessa classe com as demais do modelo clássico e do modelo quântico de computação.

Capítulo 3

Hierarquia Polinomial

Ao estudar algumas das relações entre as classes de complexidade do modelo quântico e do modelo clássico, são especialmente interessantes classes como **BPP**, **PSPACE** e as classes definidas pelos sistemas interativos de prova.

Para melhor compreender a importância de tais classes, apresentamos a hierarquia polinomial e mostramos a relação desta com as classes **BPP** e **PSPACE**.

3.1 Apresentação

Nessa seção, mostramos uma definição recursiva da hierarquia polinomial. Inicialmente, apresentamos a motivação que levou ao desenvolvimento desse conjunto de classes de complexidade.

Motivação

Um *problema de otimização* consiste na obtenção de um elemento de um domínio que maximiza ou minimiza uma função definida sobre tal domínio. Em teoria de complexidade, dá-se especial atenção aos casos em que os domínios são conjuntos discretos.

Uma maneira mais adequada de caracterizar problemas de otimização (e também as classes de complexidade) envolve o uso do conceito de completude: Dado um problema, ele é completo para quais classes de complexidade? Dada uma classe de complexidade, ela possui algum problema completo? Tais questões estimularam o estudo da *hierarquia polinomial*.

A hierarquia polinomial foi proposta inicialmente por Karp [Kar72] e formalizada por Meyer e Stockmeyer [MS73], com o objetivo de determinar classes

de complexidade para as quais alguns problemas **NP**-difíceis eram completos.

Definição

Para definir a hierarquia polinomial, utilizaremos o conceito de *oráculo*. Um oráculo sempre está associado a uma linguagem em Σ^* (e vice-versa). Uma máquina de Turing com acesso a um oráculo é uma máquina que possui uma fita extra e três estados especiais $q_?, q_y$ e q_n. Quando essa máquina está no estado $q_?$, no próximo passo ela estará no estado q_y se o conteúdo da fita extra é uma palavra que pertence à linguagem associada ao oráculo; caso contrário, o próximo estado será q_n. Logo, quando uma MT M tem acesso a um oráculo para uma linguagem L, a decisão da pertinência de uma palavra em L pode ser realizada por M em um passo.

Se $\mathbf{C}$ é uma classe de complexidade e L é uma linguagem, denotamos por $\mathbf{C}^P$ a classe de linguagens que podem ser decididas por MTs que respeitam as restrições estabelecidas para $\mathbf{C}$ e que têm acesso a um oráculo para P. Se P é uma linguagem completa para uma determinada classe $\mathbf{O}$, podemos denotar $\mathbf{C}^P$ por $\mathbf{C}^\mathbf{O}$. Dizemos nesse caso que a máquina de Turing em questão tem acesso a um oráculo para $\mathbf{O}$. Observamos que $\mathbf{co}(\mathbf{C}^\mathbf{O}) = (\mathbf{coC})^\mathbf{O}$.

A *hierarquia polinomial* é formada pelas classes $\mathbf{\Sigma}_k\mathbf{P}, \mathbf{\Pi}_k\mathbf{P}$ e $\mathbf{\Delta}_k\mathbf{P}$ para $k = 0, 1, 2, \ldots$, onde:

- $\mathbf{\Sigma}_0\mathbf{P} = \mathbf{\Pi}_0\mathbf{P} = \mathbf{\Delta}_0\mathbf{P} = \mathbf{P}$;
- $\mathbf{\Sigma}_{k+1}\mathbf{P} = \mathbf{NP}^{\mathbf{\Sigma}_k\mathbf{P}}$ para $k \geq 0$;
- $\mathbf{\Delta}_{k+1}\mathbf{P} = \mathbf{P}^{\mathbf{\Sigma}_k\mathbf{P}}$ para $k \geq 0$;
- $\mathbf{\Pi}_{k+1}\mathbf{P} = \mathbf{coNP}^{\mathbf{\Sigma}_k\mathbf{P}}$ para $k \geq 0$.

Segue da observação acima e da definição que $\mathbf{co}(\mathbf{\Sigma}_k\mathbf{P}) = \mathbf{\Pi}_k\mathbf{P}$ e que $\mathbf{co}(\mathbf{\Pi}_k\mathbf{P}) = \mathbf{\Sigma}_k\mathbf{P}$ para todo inteiro $k \geq 0$. A união de todas as classes de todos os níveis da hierarquia será denotada por $\mathbf{PH}$ (*polynomial hierarchy*).

Pela definição, o nível zero da hierarquia polinomial é a classe $\mathbf{P}$. Já o nível um envolve o uso de oráculos para $\mathbf{\Sigma}_0\mathbf{P}$. Como $\mathbf{\Sigma}_0\mathbf{P} = \mathbf{P}$ e o acesso a um oráculo para $\mathbf{P}$ não altera o poder de uma máquina de Turing (proporciona apenas uma diminuição polinomial no consumo de tempo), o nível um de $\mathbf{PH}$ é composto pelas já conhecidas classes $\mathbf{\Sigma}_1\mathbf{P} = \mathbf{NP}$, $\mathbf{\Delta}_1\mathbf{P} = \mathbf{P}$ e $\mathbf{\Pi}_1\mathbf{P} = \mathbf{coNP}$. A partir do segundo nível da hierarquia, os oráculos envolvidos passam a ter um papel importante, pois supostamente aumentam o poder das máquinas de Turing (possivelmente proporcionam diminuições super-polinomiais no consumo de tempo).

Acredita-se que, a partir do nível um, todas as classes no mesmo nível sejam distintas. Acredita-se também que a hierarquia seja própria, ou seja, que as classes de um nível estejam contidas propriamente nas classes dos níveis superiores. Vamos mostrar agora dois resultados que estabelecem relações entre classes da hierarquia.

Lema 3.1. $\mathbf{\Delta}_k\mathbf{P} \subseteq \mathbf{\Sigma}_k\mathbf{P} \cap \mathbf{\Pi}_k\mathbf{P}$ *para* $k \geq 0$.

Demonstração. Seja L uma linguagem de $\mathbf{\Delta}_k\mathbf{P}$. Por definição, existe uma MTD M com acesso a um oráculo para $\mathbf{\Sigma}_{k-1}\mathbf{P}$ que decide L em tempo polinomial.

Mas sabemos que toda MTD é uma MTND onde a relação de transição é uma função. Dessa forma, concluímos que M é uma MTND que possui uma única computação válida para uma dada entrada que verifica em tempo polinomial se uma entrada pertence ou não a L utilizando o acesso a um oráculo para $\mathbf{\Sigma}_{k-1}\mathbf{P}$. Segue daí que L é uma linguagem de $\mathbf{\Sigma}_k\mathbf{P}$ e de $\mathbf{\Pi}_k\mathbf{P}$. □

Lema 3.2. $\mathbf{\Sigma}_k\mathbf{P} \cup \mathbf{\Pi}_k\mathbf{P} \subseteq \mathbf{\Delta}_{k+1}\mathbf{P}$ *para* $k \geq 0$.

Demonstração. Seja L uma linguagem. Pela definição de $\mathbf{\Delta}_{k+1}\mathbf{P}$, existe uma MTD M com acesso a um oráculo para $\mathbf{\Sigma}_k\mathbf{P}$ que decide L em tempo polinomial.

Se $L \in \mathbf{\Sigma}_k\mathbf{P}$, então obviamente L pertence a $\mathbf{\Delta}_{k+1}\mathbf{P}$. Basta M utilizar um oráculo para a própria L. Dessa forma, com apenas uma consulta ao oráculo, M decide L.

Se $L \in \mathbf{\Pi}_k\mathbf{P}$, também é verdade que L pertence a $\mathbf{\Delta}_{k+1}\mathbf{P}$. Basta M utilizar um oráculo para $\mathbf{co}L$, que é uma linguagem de $\mathbf{\Sigma}_k\mathbf{P}$. Com apenas uma consulta ao oráculo, M decide $\mathbf{co}L$, e conseqüentemente L. □

Dessa forma, concluímos que $\mathbf{PH} = \bigcup_{k\geq 0} \mathbf{\Sigma}_k\mathbf{P} = \bigcup_{k\geq 0} \mathbf{\Pi}_k\mathbf{P} = \bigcup_{k\geq 0} \mathbf{\Delta}_k\mathbf{P}$, e que $\mathbf{coPH} = \mathbf{PH}$.

3.2 PH ⊆ PSPACE

A definição recursiva da hierarquia polinomial não é a mais conveniente para as demonstrações dos resultados que serão apresentados adiante. Por isso, apresentamos uma caracterização alternativa da hierarquia polinomial. Em seguida, mostramos algumas de suas propriedades e comentamos a relação entre as classes **PH** e **PSPACE**.

Uma caracterização alternativa

Para apresentar uma outra caracterização da hierarquia polinomial, utilizaremos os conceitos de relações polinomialmente balanceadas e polinomialmente decidíveis.

Teorema 3.3. *Seja L uma linguagem e seja $i \geq 1$. Temos que $L \in \mathbf{\Sigma}_i\mathbf{P}$ se e somente se existe uma relação R polinomialmente balanceada tal que a linguagem $L' = \{x; y : (x, y) \in R\}$ está em $\mathbf{\Pi}_{i-1}\mathbf{P}$ e $L = \{x :$ existe y tal que $(x, y) \in R\}$.*

Demonstração. O resultado será demonstrado por indução em i. Para $i = 1$, segue da definição da classe **NP** com certificados sucintos. Note que, nesse caso, a relação R, formada por duplas (x, y) onde x representa uma entrada e y um certificado sucinto da pertinência de x a uma linguagem L, além de polinomialmente balanceada, também é polinomialmente decidível. Portanto, L' está em $\mathbf{P} = \mathbf{\Pi}_0\mathbf{P}$. Vamos mostrar agora o resultado para $i > 1$.

Suponha que existe uma relação R tal que a linguagem $L' = \{x; y : (x, y) \in R\}$ está em $\mathbf{\Pi}_{i-1}\mathbf{P}$ e seja $L = \{x :$ existe y tal que $(x, y) \in R\}$. Queremos mostrar que $L \in \mathbf{\Sigma}_i\mathbf{P}$. Para isso, basta mostrar a existência de uma MTND M com acesso a um oráculo para $\mathbf{\Sigma}_{i-1}\mathbf{P}$ que decide L em tempo polinomial.

Como R é uma relação polinomialmente balanceada, existe um inteiro positivo k tal que, a partir de uma entrada x, a máquina M gera não-deterministicamente uma palavra y tal que $|y| \leq |x|^k$ e acessa o oráculo para determinar se $x; y$ pertence a L'. Mais precisamente, como o oráculo disponível para M decide $\mathbf{\Sigma}_{i-1}\mathbf{P}$ e como $\mathbf{\Pi}_{i-1}\mathbf{P} = \mathbf{co}(\mathbf{\Sigma}_{i-1}\mathbf{P})$, o oráculo será utilizado para verificar se $x; y$ pertence à linguagem $\mathbf{co}L'$.

Se para algum y a palavra $x; y$ não pertence a $\mathbf{co}L'$, então $x; y$ pertence a L', e portanto x pertence a L. Se para todo y gerado a palavra $x; y$ pertence a $\mathbf{co}L'$, então x não é palavra de L. Logo, com uma MTND com acesso a um oráculo para $\mathbf{\Sigma}_{i-1}\mathbf{P}$, podemos decidir L em tempo polinomial. Assim, temos que $L \in \mathbf{\Sigma}_i\mathbf{P}$, como queríamos.

Suponha agora que L é uma linguagem em $\mathbf{\Sigma}_i\mathbf{P}$. Devemos mostrar que existe uma relação R polinomialmente balanceada tal que a linguagem $L' = \{x; y : (x, y) \in R\}$ está em $\mathbf{\Pi}_{i-1}\mathbf{P}$ e que $L = \{x :$ existe y tal que $(x, y) \in R\}$. Como $L \in \mathbf{\Sigma}_i\mathbf{P}$, sabemos que existe uma MTND N com acesso a um oráculo para uma linguagem $\mathbf{\Sigma}_{i-1}\mathbf{P}$-completa K que decide L em tempo polinomial.

Pela hipótese de indução, como $K \in \mathbf{\Sigma}_{i-1}\mathbf{P}$, sabemos que existe uma relação S polinomialmente balanceada tal que a linguagem $L'' = \{x; y : (x, y) \in S\}$ está em $\mathbf{\Pi}_{i-2}\mathbf{P}$ e que $K = \{x :$ existe y tal que $(x, y) \in S\}$. A partir dessas informações, vamos construir a relação R desejada.

Seja $x \in \Sigma^*$ uma entrada para N. Vamos mostrar que x pertence a L se e

somente se existe uma palavra y tal que (x, y) pertence à relação R que estamos construindo. Com isso, já temos que $L = \{x : \text{existe } y \text{ tal que } (x, y) \in R\}$.

Uma tal palavra y descreve uma computação de N que aceita x como palavra de L. Se tal computação não existir, não haverá tal certificado. Como N tem acesso a um oráculo para K, alguns de seus passos consistirão em consultas a esse oráculo, que podem ter respostas "sim" ou "não". Tais informações estarão descritas em y.

Quando uma consulta descrita em y ao oráculo para K tem resposta "sim", podemos assumir que tal resposta é acompanhada por um certificado que mostra tal fato. A dupla formada pela palavra submetida à consulta e pelo certificado em questão deve ser um elemento da relação S. Logo, a checagem da validade desse passo pode ser feita em $\mathbf{\Pi}_{i-2}\mathbf{P}$.

Nos casos em que a resposta do oráculo é "não", a situação é a seguinte: N fez uma consulta ao oráculo para checar se uma palavra q pertencia à linguagem K, e a resposta foi negativa. Como $K \in \mathbf{\Sigma}_{i-1}\mathbf{P}$, a verificação da resposta do oráculo pode ser feita por meio da consulta da pertinência de q a $\mathbf{co}K$, problema este que está em $\mathbf{\Pi}_{i-1}\mathbf{P}$.

Por fim, observamos que o tamanho de y é polinomial em x. Isso porque N realiza um número polinomial de passos, e para cada consulta ao oráculo para K, as palavras armazenadas (só a palavra consultada nos casos em que a resposta é "não" e a palavra acompanhada do certificado nos casos em que a resposta é "sim") têm tamanho polinomial em $|x|$. Com isso, concluímos que R é uma relação polinomialmente balanceada.

Assim, concluímos que uma MTD polinomialmente limitada com acesso a um oráculo para uma linguagem $\mathbf{\Pi}_{i-1}\mathbf{P}$-completa pode verificar se y é um certificado para x. Ou seja, $L' = \{x; y : (x, y) \in R\} \in \mathbf{\Pi}_{i-1}\mathbf{P}$.

Assim, se $L \in \mathbf{\Sigma}_i\mathbf{P}$, então existe uma relação R polinomialmente balanceada tal que $L' = \{x; y : (x, y) \in R\} \in \mathbf{\Pi}_{i-1}\mathbf{P}$ e $L = \{x : \text{existe } y \text{ tal que } (x, y) \in R\}$, como queríamos. □

Pela simetria da hierarquia polinomial, temos o seguinte corolário.

Corolário 3.4. *Seja L uma linguagem e $i \geq 1$. Temos que $L \in \mathbf{\Pi}_i\mathbf{P}$ se e somente se existe uma relação R polinomialmente balanceada tal que a linguagem $L' = \{x; y : (x, y) \in R\}$ está em $\mathbf{\Sigma}_{i-1}\mathbf{P}$ e $L = \{x :$ para todo y com $|y| \leq |x|^k, (x, y) \in R\}$ para algum inteiro positivo k.*

Utilizando relações $(i + 1)$-árias polinomialmente balanceadas e os dois resultados anteriores, podemos definir a hierarquia polinomial de maneira não-recursiva.

Corolário 3.5. *Seja L uma linguagem e $i \geq 1$. Temos que $L \in \mathbf{\Sigma}_i\mathbf{P}$ se e somente se existe uma relação $(i+1)$-ária R polinomialmente balanceada e polinomialmente decidível tal que*

$$L = \{x : \exists y_1 \forall y_2 \exists y_3 \ldots Q_i y_i \text{ tal que } (x, y_1, \ldots, y_i) \in R\}$$

onde o j-ésimo quantificador Q_j é $\forall$ se j é par e $\exists$ se j é ímpar, para $j = 1, \ldots, i$.

Colapso da hierarquia polinomial

Nessa seção, vamos analisar a relação entre a hierarquia polinomial e a classe **PSPACE**. Mas antes, utilizando os lemas 3.1 e 3.2 e os resultados da seção anterior, vamos mostrar um fato importante.

Teorema 3.6. *Se para algum $i \geq 1$ é verdade que $\mathbf{\Sigma}_i\mathbf{P} = \mathbf{\Pi}_i\mathbf{P}$, então $\mathbf{\Sigma}_j\mathbf{P} = \mathbf{\Pi}_j\mathbf{P} = \mathbf{\Delta}_j\mathbf{P} = \mathbf{\Sigma}_i\mathbf{P}$ para todo $j > i$.*

Demonstração. Suponha que $\mathbf{\Sigma}_i\mathbf{P} = \mathbf{\Pi}_i\mathbf{P}$. Vamos mostrar inicialmente que, nesse caso, é verdade que $\mathbf{\Sigma}_{i+1}\mathbf{P} = \mathbf{\Sigma}_i\mathbf{P}$.

Seja L uma linguagem de $\mathbf{\Sigma}_{i+1}\mathbf{P}$. Pelo teorema 3.3, sabemos que existe uma relação R polinomialmente balanceada tal que a linguagem $L' = \{x; y : (x, y) \in R\}$ está em $\mathbf{\Pi}_i\mathbf{P}$ e $L = \{x : \text{existe } y \text{ tal que } (x, y) \in R\}$. Como $\mathbf{\Pi}_i\mathbf{P} = \mathbf{\Sigma}_i\mathbf{P}$, a linguagem L' também está em $\mathbf{\Sigma}_i\mathbf{P}$. Pelo teorema 3.3, sabemos que existe uma relação S polinomialmente balanceada tal que a linguagem $L'' = \{x; y : (x, y) \in S\}$ está em $\mathbf{\Pi}_{i-1}\mathbf{P}$ e $L' = \{x : \text{ existe } y \text{ tal que } (x, y) \in S\}$. Assim, temos que $(x, y) \in R$ se e somente se existe uma palavra z tal que $(x; y, z)$ pertence à relação S. Mais precisamente, $x \in L$ se e somente se existem palavras y e z tais que $(x; y, z) \in S$. A partir do teorema 3.3, concluímos que $L \in \mathbf{\Sigma}_i\mathbf{P}$.

De forma análoga, mostra-se que $\mathbf{\Pi}_{i+1}\mathbf{P} = \mathbf{\Pi}_i\mathbf{P}$. Suponha que $L \in \mathbf{\Pi}_{i+1}\mathbf{P}$. Pelo corolário 3.4, sabemos que existe uma relação R polinomialmente balanceada tal que a linguagem $L' = \{x; y : (x, y) \in R\}$ está em $\mathbf{\Sigma}_i\mathbf{P}$ e $L = \{x : \text{para todo } y \text{ com } |y| \leq |x|^k, (x, y) \in R\}$ para algum inteiro positivo k. Como $\mathbf{\Pi}_i\mathbf{P} = \mathbf{\Sigma}_i\mathbf{P}$, a linguagem L' também está em $\mathbf{\Pi}_i\mathbf{P}$. Ou seja, existe uma relação S polinomialmente balanceada tal que a linguagem $L'' = \{x; y : (x, y) \in S\}$ está em $\mathbf{\Sigma}_{i-1}\mathbf{P}$ e $L' = \{x : \text{para todo } y \text{ com } |y| \leq |x|^{k'}, (x, y) \in S\}$ para algum inteiro positivo k'. Assim, temos que $(x, y) \in R$ para todo y tal que $|y| \leq |x|^k$ se e somente se para todo z com $|z| \leq |x; y|^{k'}$ a dupla $(x; y, z)$ pertence a S. Logo, $x \in L$ se e somente se para todo par (y, z), $|y| \leq |x|^k$ e $|z| \leq |x; y|^{k'}$, $(x; y, z) \in S$. Assim, a partir do corolário 3.4, concluímos que $L \in \mathbf{\Pi}_i\mathbf{P}$.

Utilizando os lemas 3.1 e 3.2, concluímos que $\mathbf{\Delta}_{i+1}\mathbf{P} = \mathbf{\Sigma}_i\mathbf{P}$ e o teorema segue. □

Por fim, vamos analisar a relação de **PSPACE** com a hierarquia polinomial.

Teorema 3.7. $\mathbf{PH} \subseteq \mathbf{PSPACE}$.

Demonstração. Pela caracterização da hierarquia polinomial apresentada no corolário 3.5, existem certificados de pertinência de uma palavra x a uma linguagem L em **PH** que são polinomialmente balanceados e polinomialmente decidíveis.

Alguns dos componentes dos certificados estão associados ao quantificador $\exists$. Chamaremos esse conjunto de primeiro grupo. Outros estão associados ao quantificador $\forall$, que chamaremos de segundo grupo. Para cada configuração válida dos componentes do primeiro grupo, devem ser testadas todas as configurações válidas dos componentes do segundo grupo. Se para alguma configuração do primeiro grupo todas as configurações do segundo levam à aceitação, então x pertence a L. Caso contrário, x não pertence a L.

Como cada componente consome espaço polinomial, a representação de um certificado consome espaço polinomial. Logo, toda a computação pode ser realizada em espaço polinomial, contanto que após a verificação de cada certificado o espaço consumido seja reaproveitado. Dessa forma, todos os possíveis certificados podem ser gerados e verificados consumindo espaço polinomial em $|x|$. Segue desse fato que $\mathbf{PH} \subseteq \mathbf{PSPACE}$. □

Se a hierarquia polinomial for igual a **PSPACE**, então ela possuirá problemas completos, como o QBF. Mas se um problema pertence à hierarquia polinomial, então ele pertence a alguma classe de algum de seus níveis. Sendo completo, tal problema claramente induz o colapso da hierarquia, pois todos os problemas dos níveis superiores poderão ser reduzidos a tal problema. Portanto, acredita-se que $\mathbf{PH} \subset \mathbf{PSPACE}$.

3.3 $\mathbf{BPP} \subseteq \mathbf{\Sigma_2 P} \cap \mathbf{\Pi_2 P}$

Nessa seção, vamos mostrar a relação entre a classe probabilística **BPP** e as classes $\mathbf{\Sigma_2 P}$ e $\mathbf{\Pi_2 P}$, pertencentes à hierarquia polinomial. Esse resultado é importante, pois **BPP** tem um papel de destaque nas relações entre as classes de complexidade do modelo clássico e as do modelo quântico.

Teorema 3.8. $\mathbf{BPP} \subseteq \mathbf{\Sigma_2 P}$.

Demonstração. Seja L uma linguagem em **BPP**. Pela definição da classe, existe uma MTP polinomialmente limitada que decide se sua entrada pertence ou não a L corretamente com probabilidade maior ou igual que $1/2$.

Para cada entrada x de comprimento $n \geq 2$, seja $p(n)$ o tempo consumido pela MTP (os casos em que $n \leq 1$ são triviais). Vamos denotar por $A(x) \subseteq \{0,1\}^{p(n)}$ o conjunto de palavras que descrevem as seqüências de bits aleatórios que levam à aceitação de x. A i-ésima posição de cada uma dessas palavras indica o valor do bit sorteado para o i-ésimo passo da máquina. Vamos assumir que se x está em L, então $|A(x)| \geq 2^{p(n)}\left(1 - \frac{1}{2^n}\right)$, e que se x não está em L, então $|A(x)| \leq 2^{p(n)}\frac{1}{2^n}$. Ou seja, a probabilidade de a MTP devolver uma resposta errada é limitada por $\frac{1}{2^n}$. Isso pode ser garantido por meio de um número adequado de repetições de execução da MTP.

Seja $U = \{0,1\}^{p(n)}$. Vamos denotar por $\oplus : U \times U \to U$ a operação *ou exclusivo* sobre dois elementos de U. Ou seja, suponha que $x, y, z \in U$ e que $x \oplus y = z$. Denotando a i-ésima posição das palavras x, y e z por x_i, y_i e z_i, respectivamente, então $z_i = 1$ se e somente se $x_i + y_i = 1$. Dados dois elementos a e b em U, temos que $(a \oplus b) \oplus b = a$ para quaisquer a e b. Isso mostra que a operação $\oplus b$ é bijetora para todo b em U. Se b é escolhido aleatoriamente de maneira uniforme, então o elemento $a \oplus b$ é um elemento aleatório e uniformemente distribuído de U. Ou seja, $a \oplus b = u$ com probabilidade $\frac{1}{2^{p(n)}}$ para qualquer u em U.

Para t em U, considere o conjunto $A^t(x) = A(x) \oplus t = \{a \oplus t : a \in A(x)\}$. Como $\oplus t$ é bijetora, $|A^t(x)| = |A(x)|$. Vamos mostrar que se $x \in L$, podemos encontrar um conjunto relativamente pequeno $T = \{t_1, t_2, \ldots, t_k\}$ de elementos de U tal que a união dos $A^{t_i}(x)$ é igual a U (dizemos nesse caso que T *cobre* U) e que se $x \notin L$, tal conjunto não pode existir.

Sejam $t_1, t_2, \ldots, t_{p(n)}$ elementos de U sorteados aleatoriamente de maneira uniforme. Seja b um elemento de U. O conjunto formado pela seqüência sorteada *cobre* b se para algum $1 \leq i \leq p(n)$ temos que b pertence a $A^{t_i}(x)$. Como $b \in A^{t_i}(x)$ se e somente se $b \oplus t_i \in A(x)$ e como $b \oplus t_i$ é um elemento aleatório de U, se x pertence a L então a probabilidade de b não pertencer a $A^{t_j}(x)$ é limitada superiormente por 2^{-n} para $1 \leq j \leq p(n)$. Logo, a probabilidade de b não ser coberto por um conjunto $\{t_1, \ldots, t_{p(n)}\}$ é limitada por $2^{-np(n)}$. Como b é um elemento qualquer de U, tal cota vale para todo elemento de U. Como $|U| = 2^{p(n)}$, a probabilidade de existir pelo menos um elemento de U não-coberto é $2^{(1-n)p(n)} < 1$ pois $n \geq 2$. Isso mostra que existe pelo menos um conjunto $T \subseteq U$ com $p(n)$ elementos que cobre U.

Vamos analisar agora o caso em que x não pertence a L. Como $|A(x)| \leq 2^{p(n)}\frac{1}{2^n}$, não existe um conjunto T com $p(n) < 2^n$ elementos que cobre U se $x \notin L$.

Assim, podemos mostrar que $L \in \mathbf{\Sigma_2 P}$. Basta notar que L pode ser descrita da seguinte maneira

$$L = \{x : \exists T \in \{0,1\}^{p(n)^2} \text{ tal que } \forall b \in \{0,1\}^{p(n)}\, \exists j \leq p(n) \text{ tal que } b \oplus t_j \in A(x)\},$$

onde cada t_j é descrito pelos elementos de $(j-1)p(n)+1$ até $jp(n)$ de T. Pelo corolário 3.5, temos que L é uma linguagem de $\mathbf{\Sigma_2 P}$. Destacamos que o último quantificador (o segundo existencial) não altera a posição de L na hierarquia, pois verificar se para algum j, $1 \leq j \leq p(n)$, o elemento $b \oplus t_j$ está em $A(x)$ pode ser feito em tempo polinomial. □

Como **BPP** é fechada por complemento e $\mathbf{\Pi_2 P} = \mathbf{co\Sigma_2 P}$, o resultado principal da seção segue como corolário.

Corolário 3.9. $\mathbf{BPP} \subseteq \mathbf{\Sigma_2 P} \cap \mathbf{\Pi_2 P}$. □

Capítulo 4

Sistemas Interativos de Prova

Nesse capítulo, apresentamos os *sistemas interativos de prova* (**SIP**), introduzidos por Goldwasser, Micali e Rackoff [GMR85] (definição da classe **IP**) e por Babai [Bab85] (definição da classe **AM**) de maneira independente, bem como alguns de seus principais resultados.

4.1 Elementos e definições

Um **SIP** é composto por duas entidades, que denominaremos *verificador* (V) e *provador* (P). Seja Σ um alfabeto finito. Basicamente, o objetivo do sistema é decidir uma linguagem $L \subseteq \Sigma^*$. Por meio de trocas de mensagens (ou seja, através de interações), P vai tentar convencer V que $x \in L$.

Definimos V e P como máquinas de Turing. Vamos assumir que P tem poder computacional ilimitado, sendo assim capaz de computar em tempo constante qualquer função computável. Já V tem o poder de uma máquina de Turing usual.

As duas máquinas compartilham o acesso a três fitas. Em uma das fitas, P pode escrever e V só pode fazer leituras. Denominaremos tal fita F_P. Em outra fita, V pode escrever e P só pode fazer leituras. Denominaremos tal fita F_V. A outra fita contém a entrada x cuja pertinência a L está sendo verificada, e tanto P como V podem ler seu conteúdo, mas não alterá-lo.

Além dessas três fitas, um **SIP** também possui uma fita de bits aleatórios, que pode ser utilizada por V em suas computações. Denominaremos tal fita τ. Permitir que o provador leia ou não o conteúdo de τ não altera significativamente o poder de um sistema interativo, conforme veremos mais adiante, mas é a diferença entre as definições das classes **IP** e **AM**.

As trocas de mensagens entre as duas máquinas são realizadas por meio

de F_P e F_V. Como P faz todos os seus cálculos em tempo constante, vamos definir o consumo de tempo de um **SIP** como sendo o consumo de tempo de V.

Vamos definir a classe de linguagens $\mathbf{IP}(q(n))$, onde $q(n)$ é uma função de $\mathbb{N}$ em $\mathbb{N}$. Seja $L \subseteq \Sigma^*$ uma linguagem. Dizemos que L pertence à classe $\mathbf{IP}(q(n))$ se existe um sistema interativo de prova tal que para toda palavra x em Σ^*, $n = |x|$:

1. O tempo total consumido por V é polinomial em n.
2. O número de mensagens trocadas entre P e V é $O(q(n))$.
3. Se $x \in L$, então existe um provador P que sempre consegue convencer o verificador deste fato. Ou seja, V sempre vai concluir corretamente que $x \in L$.
4. Se $x \notin L$, então V vai concluir incorretamente que $x \in L$ com uma probabilidade limitada superiormente por $1/4$, independente do provador P utilizado.
5. O provador não tem acesso à fita de bits aleatórios do verificador.

Removendo a última condição da definição acima, temos a definição da classe $\mathbf{AM}(q(n))$. O seguinte resultado de Goldwasser e Sipser [GS86] mostra a relação entre as classes $\mathbf{IP}(q(n))$ e $\mathbf{AM}(q(n))$:

Teorema 4.1. $\mathbf{IP}(q(n)) = \mathbf{AM}(q(n) + 2)$. $\square$

A escolha do valor $1/4$ no terceiro item da definição de $\mathbf{IP}(q(n))$ foi arbitrária. Na verdade, qualquer constante no intervalo $(0, 1)$ pode ser utilizada. Para atingir uma determinada limitação superior constante da probabilidade de erro, basta executar o sistema em paralelo (com o mesmo provador) por um número adequado (e constante) de vezes. O mesmo vale para as classes $\mathbf{AM}(q(n))$.

A classe **IP** é definida por:

$$\mathbf{IP} = \bigcup_{k \geq 0} \mathbf{IP}(n^k).$$

Analisando as descrições dessas classes, podemos perceber que a definição da classe **NP** envolve idéias muito parecidas. Na verdade, é fácil ver que $\mathbf{NP} \subseteq \mathbf{IP}(1)$. A única troca de mensagem necessária é enviada do provador para o verificador, e consiste no certificado sucinto de pertinência usual de **NP**. Com esse certificado à disposição, o verificador pode fazer sua checagem de maneira determinística consumindo tempo polinomial, abrindo mão do uso de τ.

4.2 PSPACE = IP

Nessa seção, mostramos que **PSPACE** = **IP**, resultado esse obtido por Shamir [Sha92]. Para que algumas idéias importantes contidas nessa demonstração fiquem claras, apresentamos antes a prova de que **coNP** $\subseteq$ **IP**. Destacamos que o uso de polinômios de grau limitado e o uso de elementos aleatórios são essenciais nas duas provas.

Teorema 4.2. coNP $\subseteq$ IP.

Demonstração. Para provar essa relação, só precisamos mostrar que um problema **coNP**-completo está em **IP**. Seguindo a apresentação de Kohayakawa e Soares [KS95], mostraremos que o problema de decidir se o número cromático de um dado grafo é maior que 3 está contido em **IP**. Daqui para a frente vamos nos referir a esse problema pela sigla N3C. A completude de N3C em relação à classe **coNP** foi provada por Karp [Kar72].

A parte inicial da prova desse teorema consiste na "aritmetização" das instâncias de N3C. Esse processo corresponde à construção a partir de uma instância dada de um polinômio com propriedades relevantes para a segunda parte da prova, que consiste na descrição do protocolo de comunicação entre as máquinas.

O polinômio a ser construído utiliza como bloco básico o polinômio $p(x) = \frac{5x^2}{4} - \frac{x^4}{4}$, que tem as seguintes propriedades. Primeiro, $p(0) = 0$, e segundo, $p(x) = 1$ para $x = -2, -1, 1, 2$.

Seja G o grafo dado na instância de N3C, seja n o número de vértices de G e seja E_G o conjunto de arestas de G. Podemos assumir que $V_G = [n] = \{1, 2, \ldots, n\}$. Vamos definir o seguinte polinômio:

$$q(X_1, \ldots, X_n) = \prod_{ij \in E_G} p(X_i - X_j)$$

onde cada X_i, $1 \leq i \leq n$, pertence a um corpo finito $F = \{0, 1, \ldots, N-1\}$. A importância do valor de N ficará clara mais adiante, e portanto vamos esperar o momento oportuno para fixar seu valor. Como cada aresta de G contribui apenas com um termo no produto, o grau de $q(X_1, \ldots, X_n)$, denotado por d, é limitado superiormente por $4|E_G|$. Como o N3C é trivial quando $|E_G| = 0$, vamos assumir que $|E_G| \geq 1$.

Se, numa n-upla $x = (C_1, \ldots, C_n)$, cada C_i pertence ao conjunto $C = \{0, 1, 2\}$, então podemos considerar x como sendo a representação de uma 3-coloração de G, onde cada posição da n-upla está associada a um vértice e cada valor em C indica a cor do vértice em questão.

Uma coloração é *válida* para um grafo se e somente se dois vértices vizinhos estão com cores distintas. Logo, pelas definições de $p(X)$ e $q(X_1, \ldots, X_n)$,

temos que uma n-upla $(C_1, \ldots, C_n)$ em $\{0,1,2\}^n$ representa uma 3-coloração válida para G se e somente se $q(C_1, \ldots, C_n) = 1$. Observe que se $(C_1, \ldots, C_n)$ não é uma 3-coloração válida, então $q(C_1, \ldots, C_n) = 0$.

A partir dessa observação, concluímos que G não admite uma 3-coloração se a expressão

$$q_0 = \sum_{x_1 \in \{0,1,2\}} \sum_{x_2 \in \{0,1,2\}} \cdots \sum_{x_n \in \{0,1,2\}} q(x_1, x_2, \ldots, x_n) \tag{4.1}$$

é igual a zero.

Como o número de termos da expressão acima é exponencial em n, o verificador não pode avaliá-la em tempo polinomial de maneira trivial (ou seja, gerando todos os termos e fazendo a soma). O verificador pode entretanto interagir com um provador para estimar esse valor. O provador vai tentar convencê-lo que $q_0 = 0$. Vamos apresentar um protocolo de comunicação entre as duas máquinas e argumentar que ele obedece às condições de **IP**, mostrando assim que N3C está em **IP**.

Antes disso, vamos definir o seguinte polinômio para todo i em $[n]$:

$$q_i(X_1, \ldots, X_i) = \sum_{x_{i+1} \in \{0,1,2\}} \cdots \sum_{x_n \in \{0,1,2\}} q(X_1, \ldots, X_i, x_{i+1}, \ldots, x_n). \tag{4.2}$$

Ou seja, temos um somatório de polinômios, onde os i primeiros parâmetros são indeterminados.

A partir dessa definição, é fácil ver a seguinte relação:

$$q_{i-1}(X_1, \ldots, X_{i-1}) = \sum_{x_i \in \{0,1,2\}} q_i(X_1, \ldots, X_{i-1}, x_i). \tag{4.3}$$

Vamos apresentar agora os algoritmos do provador e do verificador.

Algoritmo do Provador

1 Cria o polinômio $\tilde{q}_0 = 0$
2 **Escreve** $\tilde{q}_0$ em F_P
3 Para i de 1 até n faça
4 Escolhe o polinômio $\tilde{q}_i(X_i)$
5 **Escreve** $\tilde{q}_i(X_i)$ em F_P
6 **Lê** de F_V um inteiro ρ_i

Algoritmo do Verificador

1 **Lê** de F_P o polinômio $\tilde{q}_0$
2 Para i de 1 até n faça
3 **Lê** de F_P o polinômio $\tilde{q}_i(X_i)$
4 Se $\tilde{q}_i(X_i)$ tem grau maior que d ou $\tilde{q}_{i-1}(\rho_{i-1}) \neq \sum_{x\in\{0,1,2\}} \tilde{q}_i(x)$
5 Rejeita a prova
6 $\rho_i \leftarrow \text{RAND}(F)$
7 **Escreve** ρ_i em F_V
8 Se $\tilde{q}_n(\rho_n) = q(\rho_1, \ldots, \rho_n)$
9 Aceita a prova
10 Senão rejeita a prova

Na primeira execução do passo 4 do algoritmo do verificador, ρ_0 está indefinido, mas como $\tilde{q}_0 = 0$, $\tilde{q}_0(\rho_0) = 0$ e portanto a condição está bem definida. Em seu passo 6, o verificador sorteia aleatoriamente de maneira uniforme um elemento do corpo F.

Vamos descrever agora a estratégia utilizada pelo provador para tentar convencer o verificador que $q_0 = 0$.

Se esse fato é verdadeiro, o provador deve ser sempre capaz de convencer o verificador. De fato, essa tarefa é simples no protocolo descrito, pois o provador só precisa escolher como $\tilde{q}_i(x)$ o polinômio $q_i(\rho_1, \ldots, \rho_{i-1}, x)$. Assim, a rejeição do passo 5 do verificador nunca acontece devido à propriedade (4.2) e porque $q_i(\rho_1, \ldots, \rho_{i-1}, x)$ tem grau limitado por d. Além disso, a igualdade do passo 8 será claramente verdadeira. Logo, sempre que $q_0 = 0$, o provador consegue convencer o verificador desse fato.

Vamos supor agora que $q_0 \neq 0$. Em sua segunda mensagem, o provador envia ao verificador um polinômio $\tilde{q}_1$ que deve ser tal que $\tilde{q}_1(0) + \tilde{q}_1(1) + \tilde{q}_1(2) = \tilde{q}_0 = 0$ e cujo grau deve ser menor ou igual a d. Do contrário, o verificador rejeita a prova de imediato. Como $q_1(0) + q_1(1) + q_1(2) = q_0 \neq 0$, claramente q_1 e $\tilde{q}_1$ são polinômios distintos. Após a checagem de $\tilde{q}_1$, o verificador sorteia um ρ_1 em F e envia-o ao provador.

Nas próximas trocas de mensagens repete-se o processo descrito. Se cada polinômio $\tilde{q}_k(x)$, $1 \leq k \leq n$, for diferente do polinômio $q_k(\rho_1, \ldots, \rho_{k-1}, x)$, no passo 8 do verificador a prova só não será rejeitada se $\tilde{q}_n(\rho_n) = q(\rho_1, \ldots, \rho_n)$, o que acontece com probabilidade limitada superiormente por $\frac{d}{N}$.

Porém, se para algum i, tivermos $\tilde{q}_{i-1}(\rho_{i-1}) = \sum_{x\in\{0,1,2\}} q_i(\rho_1, \ldots, \rho_{i-1}, x)$, então o provador pode escolher $\tilde{q}_k(x) = q_k(\rho_1, \ldots, \rho_{k-1}, x)$ para $i \leq k \leq n$. Dessa forma, ele conseguirá passar por todos os testes feitos pelo verificador e

convencê-lo que $q_0 = 0$.

Se o provador conhecesse o valor de algum ρ_i antes de escolher $\tilde{q}_i(x)$, sua tarefa seria simples. Bastaria que os polinômios $\tilde{q}_1(x), \ldots, \tilde{q}_{i-1}(x)$ fossem nulos (ou seja, sempre iguais a zero) e que $\tilde{q}_i(x)$ fosse tal que:

$$\tilde{q}_i(0) + \tilde{q}_i(1) + \tilde{q}_i(2) = 0 \tag{4.4}$$

$$\tilde{q}_i(\rho_i) = \sum_{x \in \{0,1,2\}} q_{i+1}(\rho_1, \ldots, \rho_i, x). \tag{4.5}$$

Sempre é possível construir um polinômio de grau 4 com essas características. A partir desse ponto, bastaria ao provador escolher $\tilde{q}_k(x) = q_k(\rho_1, \ldots, \rho_{k-1}, x)$ para $i + 1 \leq k \leq n$.

Porém, como a construção de $\tilde{q}_i(x)$ ocorre antes do conhecimento de ρ_i, o provador depende da sorte para que a igualdade em (4.5) ocorra. Assim, o melhor que ele pode fazer no início de cada iteração é escolher um $\tilde{q}_i(x)$ capaz de passar pelo teste da linha 4 do verificador.

Se $\tilde{q}_i(x) \neq q_i(\rho_1, \ldots, \rho_{i-1}, x)$ e os dois polinômios têm grau limitado superiormente por d, o número de valores de x para os quais eles coincidem é no máximo d. Como após a checagem do passo 4 o verificador sorteia ρ_i uniformemente em F, a probabilidade de (4.5) ocorrer é limitada superiormente por $\frac{d}{N}$.

Logo, em cada iteração, a probabilidade de o verificador sortear um ρ_i de F tal que a igualdade (4.5) ocorre pela primeira vez na iteração seguinte (ou na linha 8) é de no máximo $\left(1 - \frac{d}{N}\right)^{i-1} \frac{d}{N}$.

Dessa forma, a probabilidade de o verificador ser enganado é limitada superiormente pela soma das probabilidades de o evento acima ocorrer em uma das $n-1$ últimas iterações do Para da linha 2 do algoritmo do verificador mais a probabilidade de ocorrer igualdade apenas entre $\tilde{q}_n(\rho_n)$ e $q(\rho_1, \ldots, \rho_n)$. O valor dessa soma é limitado por

$$\sum_{i=1}^{n} \left(1 - \frac{d}{N}\right)^{i-1} \frac{d}{N} \leq \frac{dn}{N}.$$

Para que a probabilidade acima seja limitada por uma constante, vamos estabelecer um valor adequado para N. Lembrando que $d \leq 4|E_G|$ e $n = |V_G|$, se tomarmos $N = 8n^3$, temos como cota superior a constante $\frac{1}{4}$, que já é suficiente para o que queremos.

Dessa forma, temos que N3C está em **IP**, e como esse problema é completo para a classe **coNP**, temos que **coNP** $\subseteq$ **IP**. O teorema acima foi originalmente provado por Lund, Fortnow, Karloff e Nisan [LFKN92]. □

Agora passamos ao principal resultado envolvendo provas interativas.

Teorema 4.3. PSPACE = IP.

Demonstração. Para verificar que **IP** $\subseteq$ **PSPACE**, vamos descrever a simulação de um sistema interativo de prova por uma máquina de Turing determinística M que consome espaço polinomial.

Se uma determinada linguagem L está em **IP**, então existe um sistema interativo de prova para essa linguagem, e o verificador desse sistema consome tempo limitado por um polinômio $p(n)$, onde n é o tamanho da entrada.

Se o consumo de tempo do verificador é limitado por $p(n)$, então é claro que $p(n)$ também limita superiormente o número de bits aleatórios utilizados por ele, bem como o número de trocas de mensagens e a soma do comprimento de todas as mensagens que o verificador recebe do provador.

Assim, para simular esse **SIP**, M deve simular o comportamento do verificador para todas as possíveis seqüências de mensagens enviadas pelo provador. Além disso, devem ser consideradas todas as possíveis configurações da fita τ de bits aleatórios para cada uma dessas seqüências de mensagens.

Depois de realizar todas as simulações, M pára e aceita a entrada se para alguma seqüência de possíveis mensagens do provador todas as configurações de τ levam à aceitação da prova. Caso contrário, a entrada é rejeitada. Como cada uma dessas simulações pode ser feita consumindo espaço polinomial, é claro que podemos construir M de modo que o espaço por ela consumido seja polinomial.

Vamos verificar agora que **PSPACE** $\subseteq$ **IP**. Para isso, vamos mostrar que o problema QBF, que é **PSPACE**-completo, está em **IP**.

Seja $(Q_1x_1)(Q_2x_2)\ldots(Q_nx_n)B(x_1,\ldots,x_n)$ uma instância ϕ do QBF. Podemos fazer a aritmetização de ϕ da seguinte maneira. Primeiramente, substituímos em B cada expressão da forma $x \wedge y$ por xy, cada $x \vee y$ por $x \star y = x+y-xy$ e cada $\overline{x}$ por $1-x$.

Por exemplo, se tivermos $B(x_1,x_2,x_3) = x_1 \wedge (\overline{x}_2 \vee x_3) \wedge (x_2 \vee \overline{x}_3)$, a aritmetização resulta no polinômio

$$\begin{aligned} P_0(x_1,x_2,x_3) &= x_1((1-x_2)+x_3-(1-x_2)x_3)(x_2+(1-x_3)-x_2(1-x_3)) \\ &= x_1(1-x_2+x_2x_3)(1-x_3+x_2x_3) \qquad (4.6) \\ &= (x_1-x_1x_2+x_1x_2x_3)(1-x_3+x_2x_3) \\ &= x_1-x_1x_2-x_1x_3+3x_1x_2x_3-x_1x_2^2x_3-x_1x_2x_3^2+x_1x_2^2x_3^2. \end{aligned}$$

Por construção, é claro que se uma determinada valoração das variáveis faz com que $B(x_1,\ldots,x_n)$ seja verdadeira, trocando verdadeiro por 1 e falso por 0,

obtemos 1 como resposta no polinômio associado a B. Se a atribuição faz com que a fórmula fique falsa, a avaliação correspondente do polinômio será zero.

Até esse ponto, só levamos em conta a fórmula $B(x_1, \ldots, x_n)$, mas para verificar se ϕ é verdadeira, devemos levar em conta os quantificadores das variáveis. Assim, vamos introduzir transformações aritméticas no polinômio associado a $B(x_1, \ldots, x_n)$ de acordo com esses quantificadores.

Seja $P(x_1, \ldots, x_i)$ um polinômio. Definimos os seguintes polinômios a partir de P:

$$\begin{aligned}
(\forall x_i P)(x_1, \ldots, x_i) &= P(x_1, \ldots, x_{i-1}, 0)P(x_1, \ldots, x_{i-1}, 1), \\
(\exists x_i P)(x_1, \ldots, x_i) &= P(x_1, \ldots, x_{i-1}, 0) \star P(x_1, \ldots, x_{i-1}, 1), \\
(Rx_i P)(x_1, \ldots, x_i) &= P(x_1, \ldots, x_i) \mod (x_i^2 - x_i).
\end{aligned} \tag{4.7}$$

As duas primeiras expressões envolvem conversões de P em função de quantificadores e a última faz com que as potências do tipo x_i^t, para algum inteiro $t > 1$, sejam substituídas por x_i.

Seja $p_0(x_1, \ldots, x_n)$ o polinômio resultante da aritmetização de $B(x_1, \ldots, x_n)$ e sejam $p_1, p_2, \ldots, p_k$ os polinômios obtidos de p_0 por meio da aplicação da seguinte seqüência de operações:

$$\begin{aligned}
&Rx_1, Rx_2, \ldots, Rx_n, \\
&Q_n x_n, \\
&Rx_1, Rx_2, \ldots, Rx_{n-1}, \\
&Q_{n-1} x_{n-1}, \\
&\ldots \\
&Rx_1, \\
&Q_1 x_1.
\end{aligned} \tag{4.8}$$

O valor de k é $\frac{n^2+3n}{2}$ e $p_k = 1$ se ϕ é verdadeira e $p_k = 0$ caso contrário.

Para mostrar como essas transformações funcionam, vamos mostrar o que acontece quando algumas transformações são realizadas com o polinômio $P_1(x_1, x_2, x_3) = x_1 - x_1x_2 - x_1x_3 + 2x_1x_2x_3$, obtido a partir de (4.6) após operações de redução de grau em suas variáveis.

Aplicando $(\exists x_3 P)$, temos:

$$\begin{aligned}
P_2(x_1, x_2) &= (\exists x_3 P)(x_1, x_2) \\
&= (x_1 - x_1x_2 - x_1 1 + 2x_1x_2 1) + \\
&\quad (x_1 - x_1x_2 - x_1 0 + 2x_1x_2 0) - \\
&\quad (x_1 - x_1x_2 - x_1 1 + 2x_1x_2 1)(x_1 - x_1x_2 - x_1 0 + 2x_1x_2 0) \\
&= x_1 - x_1^2x_2 + x_1^2x_2^2.
\end{aligned}$$

Aplicando (Rx_1P_2), temos:

$$P_3(x_1, x_2) \;=\; (Rx_1P_2)(x_1, x_2) \;=\; x_1 - x_1x_2 + x_1x_2^2.$$

Aplicando (Rx_2P_3), temos:

$$P_4(x_1, x_2) \;=\; (Rx_2P_3)(x_1, x_2) \;=\; x_1 - x_1x_2 + x_1x_2. \tag{4.9}$$

Aplicando $(\forall x_2P_4)$, temos:

$$P_5(x_1) \;=\; (\forall x_2P_4)(x_1) \;=\; (x_1 - x_11 + x_11)(x_1 - x_10 + x_10) = x_1^2.$$

Por fim, aplicando (Rx_1P_5), temos:

$$P_6(x_1) \;=\; (Rx_1P_5)(x_1) \;=\; x_1.$$

Pela descrição da seqüência, falta aplicar a transformação Q_1x_1. Se esta for $\forall x_1$, a resposta será zero e ϕ é falsa. Se esta for $\exists x_1$, a resposta será um e ϕ é verdadeira.

A idéia geral do **SIP** que vamos descrever é a seguinte. O provador, para convencer o verificador que ϕ é verdadeira, envia para ele seqüencialmente polinômios $\tilde{p}_k, \tilde{p}_{k-1}, \ldots, \tilde{p}_0$ em uma variável.

O provador tenta convencer o verificador de que esses polinômios nada mais são do que os polinômios $p_k, \ldots, p_0$ parcialmente calculados (com todas as suas variáveis exceto uma substituídas por valores fornecidos pelo verificador).

O verificador, de sua parte, testa a consistência entre os polinômios recebidos do provador e a consistência deles contra a seqüência de operações que os originou. Além disso, o verificador sorteia em um corpo F valores para serem utilizados no lugar das variáveis na transformação dos polinômios $\tilde{p}_k, \ldots, \tilde{p}_0$ em polinômios de uma única variável.

Denotando por $o_1, \ldots, o_k$ as operações da lista de transformações (4.8) do polinômio original, temos que $p_i = o_i(p_{i-1})$. Vamos apresentar agora os algoritmos do provador e do verificador.

Algoritmo do Provador

1 $j \leftarrow 1$
2 Para $i = k-1, \ldots, 0$ faça
3 Escolhe o polinômio $\tilde{p}_i(x)$
4 **Escreve** $\tilde{p}_i(x)$ em F_P
5 **Lê** de F_V um inteiro c_j

Algoritmo do Verificador

1 $e = 1$
2 Para i de $k-1$ até 0 faça
3 **Lê** de F_P o polinômio $\tilde{p}_i(x)$
4 Se o grau de $\tilde{p}_i(x_j)$ excede d então
5 Rejeita a prova
6 Se $o_{i+1} = Rx_j$ para algum j então
7 Se $(Rx_j\tilde{p}_i)(c_j) = \tilde{p}_i(0) + (\tilde{p}_i(1) - \tilde{p}_i(0))c_j \neq e$ então
8 Rejeita a prova
9 Se $o_{i+1} = \exists x_j$ para algum j então
10 Se $(\exists x_j\tilde{p}_i) = \tilde{p}_i(0) + \tilde{p}_i(1) - \tilde{p}_i(0)\tilde{p}_i(1) \neq e$ então
11 Rejeita a prova
12 Se $o_{i+1} = \forall Rx_j$ para algum j então
13 Se $(\forall x_j\tilde{p}_i) = \tilde{p}_i(0)\tilde{p}_i(1) \neq e$ então
14 Rejeita a prova
15 $c_j \leftarrow \text{RAND}(F)$
16 $e \leftarrow \tilde{p}_i(c_j)$
17 **Escreve** c_j em F_V
18 Se $p_0(c_1, c_2, \ldots, c_n) \neq e$ então
19 Rejeita a prova
20 Aceita a prova

O valor de d, que aparece no passo 4 do verificador, será fixado posteriormente no texto. Assumimos que o valor de j é conhecido pelo provador e pelo verificador (ou seja, assumimos que eles "compartilham" essa variável).

Assim como no caso da prova de que **coNP** $\subseteq$ **IP**, a interação com o provador é importante para o verificador, pois ele não é capaz de decidir se ϕ é ou não verdadeira de maneira trivial em tempo polinomial no tamanho da entrada (ao realizar as operações de (4.8), os polinômios obtidos podem ter uma quantidade de monômios exponencial no número de variáveis da instância de QBF).

Agora, vamos explicar o protocolo de interação entre o provador e o verificador. Conforme já destacamos, o provador vai tentar convencer o verificador que o valor final produzido após as transformações é 1.

Nesse protocolo, o polinômio original (a aritmetização de $B(x_1, \ldots, x_n)$) será reconstruído seguindo a ordem inversa à utilizada para avaliar se ϕ é verdadeira. Ou seja, a partir do polinômio $p_k = c$, onde c é uma constante, são

realizadas transformações que aumentam o número de variáveis ou alteram o grau dos monômios envolvidos, de modo que no final o polinômio produzido é o polinômio associado à aritmetização de $B(x_1, \ldots, x_n)$.

Porém, como já comentamos, o número de termos dos polinômios pode não ser polinomial no tamanho da entrada. Para contornar tal adversidade, o verificador mantém os polinômios em função de uma única variável (a variável x_j, onde j é obtido pelo provador entre as linhas 6 e 14 de seu algoritmo). As outras variáveis são substituídas por valores aleatórios de um corpo finito F. Isso vai ajudá-lo a manter a polinomialidade no tempo de execução sem impedi-lo de, com alta probabilidade, detectar eventuais truques do provador.

A cada passo, o provador realiza uma das três transformações de 4.7. Já o verificador checa a consistência dos polinômios enviados pelo provador e sorteia aleatoriamente de maneira uniforme um valor de F para x_j, procurando dessa forma reduzir a probabilidade de o provador ludibriá-lo. Uma mesma variável pode receber valores de F diversas vezes durante a execução do protocolo.

Por motivos de clareza, vamos explicar com mais detalhes o protocolo por meio de um exemplo. Esse exemplo será basicamente a descrição da interação entre o provador e o verificador para a instância que estamos considerando nessa prova. Vamos mostrar como essa interação vai funcionar no caso em que $(Q_1 x_1) = (\exists x_1 P)$ (ou seja, no caso em que ϕ é verdadeira e o provador consegue convencer o verificador desse fato). Em seguida vamos comentar o caso em que ϕ é falsa.

Durante as interações, o provador fornece polinômios para o verificador, que armazena tanto suas descrições como o valor que eles produzem quando suas variáveis são substituídas pelos valores aleatórios sorteados durante a interação. O valor produzido pelo último polinômio será denotado por e, que inicialmente é igual a 1.

Inicialmente, temos a seguinte interação:

- Provador **envia** $\tilde{p}_6(x_1) = x_1$.
- Verificador confirma que $(\exists x_1 \tilde{p}_6) = 0 + 1 - 0.1 = 1 = e$.
- Verificador sorteia um valor para c_1 em F.
- Verificador atualiza $e \leftarrow \tilde{p}_6(c_1)$.
- Verificador **envia** c_1 para o provador.

O primeiro polinômio que o provador envia para o verificador é justamente o penúltimo polinômio produzido na avaliação da instância. Esse fato não é coincidência. Como ϕ é verdadeira, bastará ao provador enviar os polinômios

obtidos naquela simulação, com as devidas trocas de variáveis pelos valores sorteados pelo verificador, para que ele seja capaz de passar nos testes realizados, conforme veremos a seguir.

A primeira checagem do verificador é testar a transformação do polinômio recebido com a expressão $(\exists x_1 \tilde{p}_6)$ para saber se o resultado produzido é igual a e (que nessa primeira iteração é 1). Se isso não fosse verdade ou se o grau de $\tilde{p}_6$ fosse maior que d, a prova seria rejeitada. Esse não é o caso nesse exemplo.

Como o polinômio enviado pelo provador está em função de x_1, no final da iteração o verificador sorteia aleatoriamente com distribuição uniforme um valor c_1 no corpo F para x_1. Nas interações seguintes, toda vez que x_1 precisar ser substituído por uma constante, esse será o valor utilizado, até o momento em que um novo valor para essa variável seja sorteado.

Além disso, no final de cada interação, um novo valor de e é calculado. Ele vai conter o valor do polinômio enviado pelo provador calculado no ponto sorteado nesta interação. Assim, na próxima iteração, o valor de e será $\tilde{p}_6(c_1)$.

A segunda interação é dada abaixo:

- Provador **envia** $\tilde{p}_5(x_1) = x_1^2$.
- Verificador confirma que $(Rx_1\tilde{p}_5)(c_1) = 0 + (1-0)c_1 = c_1 = e$.
- Verificador sorteia um novo valor para c_1 em F.
- Verificador atualiza $e \leftarrow \tilde{p}_5(c_1)$.
- Verificador **envia** c_1 para o provador.

Nessa interação, nenhuma variável foi introduzida. Porém, o polinômio foi alterado. Para checar a consistência, o verificador analisa se $\tilde{p}_5$ transformado com (Rx_1) e aplicado no c_1 do começo desta fase é igual a e. Além disso, o valor de c_1 foi alterado (novamente foi sorteado aleatoriamente com distribuição uniforme), bem como o de e.

Vamos agora para a próxima interação:

- Provador **envia** $\tilde{p}_4(x_2) = c_1$.
- Verificador certifica que $(Ax_2\tilde{p}_4)(c_1) = c_1c_1 = e$.
- Verificador sorteia um valor para c_2 em F.
- Verificador atualiza $e \leftarrow \tilde{p}_4(c_2)$.
- Verificador **envia** c_2 para o provador.

Aqui, como no primeiro passo, uma nova variável foi introduzida. Assim, a variável x_1 foi trocada nos polinômios envolvidos por c_1, e $\tilde{p}_4$ está em função de x_2. Ou seja, $\tilde{p}_4(x_2) = P_4(c_1, x_2)$, onde P_4 é o polinômio descrito em (4.9). Conseqüentemente, o valor sorteado no final dessa interação servirá para substituir x_2 posteriormente.

Nessas interações, vimos o funcionamento do protocolo no caso das três transformações possíveis. As demais transformações seguem a ordem inversa da utilizada na avaliação da instância, e as interações são semelhantes às que vimos.

Como nesse caso ϕ é verdadeira, o provador será capaz de convencer o verificador que de fato o resultado é 1. A cada passo, o polinômio enviado será justamente o polinômio obtido na realização do processo inverso, com as variáveis trocadas pelas constantes sorteadas, e no final o verificador acabará aceitando o argumento do provador. A seqüência de polinômios obtida a partir do processo inverso para uma determinada instância é chamada *seqüência correta de polinômios*.

Se a instância não é verdadeira, porém, o provador não pode utilizar a seqüência correta desde o começo, pois a resposta é zero e certamente a prova será rejeitada na primeira checagem de consistência do verificador. Para tentar enganar o verificador, o melhor que o provador pode fazer é escolher polinômios para passar nos testes do verificador, e torcer para que em algum momento o verificador sorteie um valor que possibilite o uso da seqüência correta de polinômios no protocolo. Se isso acontecer, ele será capaz de ludibriá-lo.

Como os valores escolhidos pelo verificador para as variáveis são sorteados aleatoriamente e o grau dos polinômios envolvidos sempre é limitado superiormente por d, a probabilidade de ocorrência de tal evento em cada iteração é limitada superiormente por $\frac{d}{|F|}$. Ou seja, de certa forma chegamos numa situação semelhante à encontrada na demonstração do resultado anterior.

O grau d dos polinômios envolvidos será limitado superiormente por n, onde n é o número de variáveis da fórmula. Como o número de transformações realizadas é quadrático em n, a probabilidade de o verificador ser ludibriado é dada por:

$$\sum_{i=1}^{\frac{n^2+3n}{2}} \left(1 - \frac{d}{|F|}\right)^{i-1} \frac{d}{|F|} \leq \frac{d(\frac{n^2+3n}{2})}{|F|}.$$

Assim, escolhendo um corpo F de tamanho $4n^3+4n$, temos que a probabilidade de o verificador aceitar erroneamente uma demonstração é limitada superiormente pela constante $\frac{1}{4}$, como queríamos. □

Capítulo 5

Modelo Quântico de Computação

Nesse capítulo, apresentamos as definições básicas do modelo quântico que permitirão a compreensão dos resultados que serão estudados.

5.1 Preliminares

Nessa seção, apresentamos as ferramentas matemáticas mais utilizadas nos estudos do modelo quântico de computação.

Espaços de Hilbert

Na computação quântica, o conceito de espaço de Hilbert tem grande importância. Um espaço de Hilbert é um espaço vetorial definido sobre o conjunto dos números complexos com algumas propriedades que descrevemos a seguir. Nesse texto, todos os espaços têm dimensão finita. Denotamos o *conjugado* de um número complexo x por x^* (ou seja, se $x = a + bi$, então $x^* = a - bi$)..

Quando falarmos de um elemento ϕ de um espaço vetorial, vamos utilizar $|\phi\rangle$ para denotar sua representação como vetor coluna e $\langle\phi|$ para denotar o transposto conjugado de $|\phi\rangle$.

Seja H um espaço vetorial sobre os complexos. Uma função $f : H \times H \to \mathbb{C}$ tal que, para quaisquer $\phi, \phi', \psi \in H$ e quaisquer $a, b \in \mathbb{C}$,

- $f(\psi, \phi) = f(\phi, \psi)^*$;
- $f(\psi, \psi) \geq 0$, com $f(\psi, \psi) = 0$ se e somente se $\psi = 0$;
- $f(\psi, a\phi + b\phi') = af(\psi, \phi) + bf(\psi, \phi')$;

é chamada de *produto interno* em H. Em particular, $f(\psi, \phi) = \langle\psi||\phi\rangle$ é um produto interno em H. Denotamos $\langle\psi||\phi\rangle$ simplesmente por $\langle\psi|\phi\rangle$. Se $H = \mathbb{C}^n$, o produto interno $\langle\cdot,\cdot\rangle$ entre dois elementos $x = (x_1, \ldots, x_n)$ e $y = (y_1, \ldots, y_n)$ de H fica

$$\big\langle (x_1, \ldots, x_n) \big| (y_1, \ldots, y_n) \big\rangle = \sum_{i=1}^{n} x_i^* y_i,$$

e falamos sobre o *espaço n-dimensional com produto interno complexo.*

O produto interno induz uma norma de maneira bem natural:

$$\|x\| = \sqrt{\langle x|x\rangle}.$$

Dizemos que um espaço vetorial H é *completo* se para cada seqüência de vetores $x_1, x_2, \ldots$ em H tal que

$$\lim_{m,n\to\infty} \|x_m - x_n\| = 0,$$

existe um vetor x de H tal que

$$\lim_{n\to\infty} \|x_n - x\| = 0.$$

Um *espaço de Hilbert* é um espaço vetorial completo com produto interno complexo. Todos os espaços de Hilbert n-dimensionais são isomorfos, e portanto podemos denotar qualquer um desses espaços por $\mathcal{H}_n$.

Um *subespaço* H' de um espaço de Hilbert H é um subconjunto de H fechado sobre as operações de soma e de produto escalar. Qualquer subespaço de um espaço de Hilbert de dimensão finita (ou seja, todos os espaços considerados nesse texto) também é um espaço de Hilbert.

Uma propriedade importante dos espaços de Hilbert diz respeito à sua decomposição em subespaços ortogonais, conforme descreve o seguinte teorema, apresentado em [Gru99]:

Teorema 5.1. *Para cada subespaço fechado W de um espaço de Hilbert H, existe um único subespaço $W^\perp = \{|\phi\rangle \mid \langle\phi|\psi\rangle = 0$ para qualquer $|\psi\rangle \in W\}$ tal que cada $|\phi\rangle \in H$ tem uma única representação $|\phi\rangle = |\psi_1\rangle + |\psi_2\rangle$, com $|\psi_1\rangle \in W$ e $|\psi_2\rangle \in W^\perp$. Nesse caso, escrevemos $H = W \oplus W^\perp$.* □

Discutiremos adiante a importância em computação quântica dos espaços de Hilbert cuja dimensão é uma potência de 2, bem como as bases ortonormais que adotaremos para esses espaços.

Produto tensorial

Sejam $\mathcal{H}_n, \mathcal{H}_m$ espaços de Hilbert gerados por bases ortonormais $B_n = \{|\phi_1\rangle, \ldots, |\phi_n\rangle\}$ e $B_m = \{|\psi_1\rangle, \ldots, |\psi_m\rangle\}$, respectivamente. O *produto tensorial* de $\mathcal{H}_n$ e $\mathcal{H}_m$, denotado por $\mathcal{H}_n \otimes \mathcal{H}_m$, é um espaço vetorial gerado por

$$B = B_n \times B_m = \Big\{ \big(|\phi_i\rangle, |\psi_j\rangle\big) : |\phi_i\rangle \in B_n, |\psi_j\rangle \in B_m \Big\},$$

e com produto interno entre $\big(|\phi_i\rangle, |\psi_j\rangle\big)$ e $\big(|\phi_k\rangle, |\psi_l\rangle\big)$ dado por $\langle\phi_i|\phi_k\rangle\langle\psi_j|\psi_l\rangle$ para quaisquer $|\phi_i\rangle, |\phi_k\rangle \in B_n$ e $|\psi_j\rangle, |\psi_l\rangle \in B_m$. Por essa definição, notamos que B é um conjunto independente com nm elementos ortonormais que gera um espaço nm-dimensional.

Sejam $\mathcal{H}_n$, $\mathcal{H}_m$, B_n e B_m como na definição acima, $|\phi\rangle = \sum_{i=1}^{n} c_i|\phi_i\rangle \in \mathcal{H}_n$ e $|\psi\rangle = \sum_{j=1}^{m} d_j|\psi_j\rangle \in \mathcal{H}_m$. O *produto tensorial* de $|\phi\rangle$ e $|\psi\rangle$ é um vetor do espaço $\mathcal{H}_n \otimes \mathcal{H}_m$, dado por

$$|\phi\rangle \otimes |\psi\rangle = \sum_{i=1}^{n}\sum_{j=1}^{m} c_i d_j \big(|\phi_i\rangle, |\psi_j\rangle\big).$$

Eventualmente escreveremos $|\phi\rangle|\psi\rangle$ no lugar de $|\phi\rangle \otimes |\psi\rangle$.

A partir dessa definição de produto interno, temos que $\mathcal{H}_n \otimes \mathcal{H}_m$ é um espaço com produto interno complexo. Além disso, esse espaço também é completo. Portanto, temos que $\mathcal{H}_n \otimes \mathcal{H}_m$ é um espaço de Hilbert, isomorfo a $\mathcal{H}_{nm}$.

Por fim, vamos definir o produto tensorial para matrizes. Isso será importante mais adiante, quando matrizes representarão operadores lineares nos espaços de Hilbert. Considere as matrizes

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \text{ e } B = \begin{pmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mm} \end{pmatrix}.$$

O *produto tensorial de A e B*, denotado por $A \otimes B$, é definido por

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{n1}B & \cdots & a_{nn}B \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}b_{11} & \cdots & a_{11}b_{1m} & & a_{1n}b_{11} & \cdots & a_{1n}b_{1m} \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ a_{11}b_{m1} & \cdots & a_{11}b_{mm} & & a_{1n}b_{m1} & \cdots & a_{1n}b_{mm} \\ & \vdots & & & & \vdots & \\ a_{1n}b_{11} & \cdots & a_{1n}b_{1m} & & a_{nn}b_{11} & \cdots & a_{nn}b_{1m} \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ a_{1n}b_{m1} & \cdots & a_{1n}b_{mm} & & a_{nn}b_{m1} & \cdots & a_{nn}b_{mm} \end{pmatrix}.$$

Bases ortonormais padrão

Utilizando o produto tensorial de vetores, vamos definir uma base ortonormal para cada espaço de Hilbert cuja dimensão é uma potência de 2.

Sejam $B_1, \ldots, B_m$ bases ortonormais dos espaços de Hilbert $\mathcal{H}_1, \ldots, \mathcal{H}_m$. Então o conjunto

$$B_1 \otimes \cdots \otimes B_m = \bigotimes_{i=1}^{m} B_i = \{|\phi_1\rangle \otimes \cdots \otimes |\phi_m\rangle : |\phi_i\rangle \in B_i\}$$

é uma base para o espaço de Hilbert

$$\mathcal{H}_1 \otimes \cdots \otimes \mathcal{H}_m = \bigotimes_{i=1}^{m} \mathcal{H}_i.$$

Para o espaço de Hilbert $\mathcal{H}_2$ utilizaremos como base o conjunto $B_2 = \{|0\rangle, |1\rangle\}$, onde

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ e } |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Já o conjunto

$$\bigotimes_{i=1}^{n} B_2 = \left\{|x_1\rangle \otimes \cdots \otimes |x_n\rangle : x_1 \cdots x_n \in \{0,1\}^n\right\}$$

será a base ortonormal que utilizaremos para o espaço de Hilbert

$$\mathcal{H}_{2^n} = \bigotimes_{i=1}^{n} \mathcal{H}_2,$$

de dimensão 2^n. Para facilitar a notação, representaremos tal base por $\{|x\rangle : x \in \{0,1\}^n\}$.

Projeções

Dentre os operadores aplicáveis nos elementos de um espaço de Hilbert, têm especial importância as projeções. Se $H = W \oplus W^\perp$ é uma decomposição de um espaço de Hilbert H em dois subespaços ortogonais W e $W^\perp$ e $|\phi\rangle \in H$ é representado por $|\phi\rangle = |\phi_W\rangle + |\phi_{W^\perp}\rangle$, onde $|\phi_W\rangle \in W$ e $|\phi_{W^\perp}\rangle \in W^\perp$, o operador

$$P_W(|\phi\rangle) = |\phi_W\rangle$$

é uma *projeção* no subespaço W. Nesse texto, daremos especial atenção às projeções em subespaços gerados por subconjuntos não-vazios da base ortonormal padrão do espaço $\mathcal{H}_n$, onde n é uma potência de 2. Vamos analisar inicialmente projeções em espaços gerados por um único vetor de $\mathcal{H}_2$.

Seja $\mathcal{H}_{|0\rangle}$ o espaço vetorial gerado pelo vetor $|0\rangle$, e seja $\mathcal{H}_{|1\rangle}$ o espaço vetorial gerado pelo vetor $|1\rangle$. É claro que $\mathcal{H}_2 = \mathcal{H}_{|0\rangle} \oplus \mathcal{H}_{|1\rangle}$. A projeção no subespaço $\mathcal{H}_{|0\rangle}$, denotada por $P_{|0\rangle}$, é a matriz dada por $|0\rangle\langle 0|$, e a projeção no subespaço $\mathcal{H}_{|1\rangle}$ é a matriz $P_{|1\rangle} = |1\rangle\langle 1|$.

Vamos analisar agora projeções em vetores de $\mathcal{H}_{2^n}$. Seja $x \in \{0,1\}^n$ e $S \subseteq [n]$. Denotamos por x_S a seqüência de tamanho $|S|$ formada pelas coordenadas de x indexadas pelos elementos de S. Para cada $y \in \{0,1\}^{|S|}$, denotamos por $\mathcal{H}(S,y)$ o espaço vetorial gerado por $\{|x\rangle : x \in \{0,1\}^n \text{ e } |x_S\rangle = |y\rangle\}$, que é um subconjunto da base do espaço $\mathcal{H}_{2^n}$.

De maneira análoga a que ocorria com $\mathcal{H}_2$, para cada $y \in \{0,1\}^{|S|}$, existe uma projeção $P(S,y)$ associada, dada por

$$P(S,y) = \sum_{\substack{x:x\in\{0,1\}^n \\ |x_s\rangle=|y\rangle}} |x\rangle\langle x|.$$

Além disso, temos que os subespaços $\mathcal{H}(S,y)$ são dois a dois ortogonais para um S fixo. Dessa forma, temos que todo $|\phi\rangle$ em $\mathcal{H}_{2^n}$ possui uma única representação $|\phi\rangle = \sum_{y\in\{0,1\}^{|S|}} |\phi_y\rangle$, onde $|\phi_y\rangle \in \mathcal{H}(S,y)$ para cada y em $\{0,1\}^{|S|}$ é o resultado da aplicação de $P(S,y)$ no vetor $|\phi\rangle$. Assim, temos também que $\mathcal{H}_{2^n} = \bigoplus_{y\in\{0,1\}^{|S|}} \mathcal{H}(S,y)$.

Uma propriedade importante das projeções é a idempotência: se P é uma projeção e $|\phi\rangle$ é um vetor de $\mathcal{H}_n$, então $P|\phi\rangle = P^2|\phi\rangle$. Além disso, temos também que as projeções são transformações *auto-adjuntas*, ou seja, a matriz que representa uma projeção P é igual à sua transposta conjugada.

5.2 Bits e registradores quânticos

Um *qubit* ou *bit quântico* é um vetor unitário em $\mathcal{H}_2$, isto é, um vetor $|\phi\rangle \in \mathcal{H}_2$ é um qubit se

$$|\phi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle, \tag{5.1}$$

com $\alpha_0, \alpha_1 \in \mathbb{C}$ e $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Dizemos que os vetores $|0\rangle$ e $|1\rangle$ são os *estados básicos* e que o qubit $|\phi\rangle$ está numa *superposição* de estados básicos. Chamamos o coeficiente complexo α_j de *amplitude* do estado básico $|j\rangle$, para $j = 0, 1$.

Um *registrador quântico* de n qubits é um vetor unitário em $\mathcal{H}_{2^n}$, isto é, um vetor $|\phi\rangle \in \mathcal{H}_{2^n}$ é um registrador quântico de n qubits se

$$|\phi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle, \tag{5.2}$$

com $\alpha_x \in \mathbb{C}$ para todo $x \in \{0,1\}^n$ e $\sum_{x \in \{0,1\}^n} |\alpha_x|^2 = 1$.

As nomenclaturas de qubits se estendem para os registradores quânticos. Os estados $|x\rangle$ com $x \in \{0,1\}^n$ são os *estados básicos*, o registrador $|\phi\rangle$ é dito uma *superposição* de estados básicos e o coeficiente complexo α_x é chamado de *amplitude* do estado básico $|x\rangle$ para todo $x \in \{0,1\}^n$.

Algumas vezes, será conveniente expressarmos o estado (5.2) como

$$|\phi\rangle = \sum_{x=0}^{2^n-1} \alpha_x |x\rangle, \tag{5.3}$$

onde estamos substituindo as cadeias de caracteres de $\{0,1\}^n$ pelos valores numéricos que essas cadeias representam, se interpretadas como representações binárias de números.

Por fim, tanto no caso dos qubits como dos registradores, eventualmente utilizamos o termo *estado puro* para nos referirmos a uma superposição.

5.3 Medições

Ao contrário do que acontece com os bits clássicos, a leitura de um qubit pode alterá-lo. Em computação quântica, a leitura de qubits e registradores é denominada *medição*. Uma medição é um experimento probabilístico envolvendo projeções.

Seja $|\phi\rangle$ um registrador quântico de n qubits e S um subconjunto de $[n]$. Chamamos de *medição de* $|\phi\rangle$ *nos qubits de* S o evento probabilístico que escolhe,

dentre as projeções $\{P(S,y) : y \in \{0,1\}^{|S|}\}$, a projeção $P(S,y)$ para um certo y com probabilidade

$$\|P(S,y)(|\phi\rangle)\|,$$

e aplica tal projeção a $|\phi\rangle$. O resultado da medição é o y, que corresponde à projeção escolhida. Note que, somando as probabilidades para cada possível configuração de y, temos sempre 1 como resposta.

Por fim, destacamos que a probabilidade de um determinado y ser obtido após uma medição de $|\phi\rangle$ nos qubits de S também é dada por

$$p(y) = \langle\phi|P(S,y)|\phi\rangle.$$

5.4 Entrelaçamento quântico

Considere um registrador $|\psi\rangle$ de 2 qubits no estado

$$|\psi\rangle = \frac{1}{\sqrt{2}}\big(|00\rangle + |11\rangle\big). \tag{5.4}$$

Se fizermos uma medição de $|\psi\rangle$ no primeiro qubit, obteremos 0 com probabilidade $1/2$ e 1 com probabilidade $1/2$. No primeiro caso, $|\psi\rangle$ passa a valer $|00\rangle$, e no segundo $|11\rangle$. Com isso, uma posterior medição no segundo qubit já tem seu resultado determinado, isto é, com probabilidade 1 obteremos 0 no primeiro caso e 1 no segundo. Portanto, as medições nos qubits de $|\psi\rangle$ não são independentes.

Isso não ocorreria, entretanto, se $|\psi\rangle$ estivesse no estado

$$|\psi\rangle = \frac{1}{2}\big(|00\rangle + |01\rangle + |10\rangle + |11\rangle\big).$$

Nesse caso, o resultado de uma medição do primeiro qubit não afetaria as probabilidades na medição do segundo qubit realizada posteriormente.

Seja $|\phi\rangle$ um registrador de n qubits. Se $|\phi\rangle$ não pode ser escrito na forma $|\phi\rangle = \bigotimes_{i=1}^{n} |\phi_i\rangle$, onde $|\phi_i\rangle$ é um qubit para cada $1 \leq i \leq n$, então dizemos que $|\phi\rangle$ está num *estado entrelaçado* (*entangled state*).

Em geral, se um estado $|\phi\rangle$ pode ser escrito como o produto tensorial de n qubits, então estes "manterão sua independência". Suponha que $|\phi_1\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$, $|\phi_2\rangle = \beta_0|0\rangle + \beta_1|1\rangle$ e que $|\phi\rangle = |\phi_1\rangle \otimes |\phi_2\rangle = \alpha_0\beta_0|00\rangle + \alpha_0\beta_1|01\rangle + \alpha_1\beta_0|10\rangle + \alpha_1\beta_1|11\rangle$. Se medirmos $|\phi\rangle$ no primeiro qubit com relação à base padrão, obtemos 0 com probabilidade $|\alpha_0\beta_0|^2 + |\alpha_0\beta_1|^2 = |\alpha_0|^2$ e o novo estado de $|\phi\rangle$ será $|0\rangle \otimes |\phi_2\rangle$; e obtemos 1 com probabilidade $|\alpha_1\beta_0|^2 + |\alpha_1\beta_1|^2 = |\alpha_1|^2$ e $|\phi\rangle$ passa a ser $|1\rangle \otimes |\phi_2\rangle$.

A existência de estados quânticos entrelaçados é a principal razão pela qual computadores quânticos não podem ser facilmente simulados eficientemente por computadores clássicos (mesmo no modelo probabilístico). De fato, se todo registrador quântico com n qubits pudesse sempre ser escrito como o produto tensorial de n qubits, então bastaria armazenar classicamente $2n$ números complexos por registrador quântico. Entretanto, é uma conseqüência da existência de estados entrelaçados que nem sempre podemos dividir um estado quântico em suas "partes componentes" e operar nestas separadamente. Assim, para armazenar classicamente o estado de um registrador quântico de n qubits, parece ser necessário armazenar 2^n números complexos.

5.5 Circuitos

Nessa seção, apresentamos o modelo de computação quântico baseado em circuitos. Iniciamos com uma discussão sobre circuitos no modelo clássico, e em seguida introduzimos os circuitos quânticos.

5.5.1 Circuitos booleanos

Seja F_2 o corpo com dois elementos, 0 e 1. Podemos dizer de maneira sucinta que circuitos booleanos são dispositivos utilizados para o cálculo de funções de domínio F_2^n e imagem F_2^m, onde n e m são inteiros positivos. Chamaremos as funções com essas características de *funções booleanas*. Os circuitos booleanos são compostos por elementos denominados *portas lógicas*, que nada mais são do que funções booleanas que têm domínio e imagem de tamanho constante. Um exemplo de porta lógica é a porta *fanout*, que é uma função $f : F^2 \to F_2^2$, onde $f(x) = (x, x)$.

Seja $f : F_2^n \to F_2^m$ uma função booleana e C um conjunto de portas lógicas. Seja $(x_1, \ldots, x_n)$ um elemento arbitrário de F_2^n e $(y_1, \ldots, y_m) = f(x_1, \ldots, x_n)$. Um *circuito booleano* para f é um grafo dirigido acíclico conexo onde cada vértice é rotulado ou com x_i, para $1 \leq i \leq n$ (as chamadas variáveis de entrada), ou com y_j, para $1 \leq j \leq m$ (as chamadas variáveis de saída), ou com uma porta de C. Um vértice rotulado com uma variável de entrada possui grau de entrada zero, um vértice rotulado com uma variável de saída possui grau de saída zero e grau de entrada um e um vértice rotulado com uma porta lógica de domínio F_2^a e imagem F_2^b tem grau de entrada e de saída a e b respectivamente. Os arcos desse grafo correspondem aos *fios* do circuito booleano. A *complexidade* de um circuito booleano é o número de vértices desse grafo. O circuito booleano para f deve ser tal que, se ele recebe em suas variáveis de entrada elementos $x_1, \ldots, x_n$

de F_2, então suas variáveis de saída $y_1, \ldots, y_m$ vão conter no final o elemento de F_2^m tal que $f(x_1, \ldots, x_n) = (y_1, \ldots, y_m)$.

Um conjunto de portas lógicas que possibilita a representação de todas as funções booleanas como grafos dirigidos acíclicos conexos é denominado *conjunto universal*. O conjunto universal que utilizaremos nessa seção será composto pelas seguintes funções: $\wedge : F_2^2 \to F_2$ (a porta lógica *and*), onde $\wedge(x_1, x_2) = 1$ se e somente se $x_1 = x_2 = 1$; $\vee : F_2^2 \to F_2$ (a porta lógica *or*), onde $\vee(x_1, x_2) = 0$ se e somente se $x_1 = x_2 = 0$; e $\neg : F_2 \to F_2$ (a porta lógica *not*), onde $\neg x = 1 - x$.

Um resultado bem conhecido mostra que todas as funções booleanas podem ser computadas por circuitos booleanos compostos apenas por portas lógicas $\wedge$, $\vee$ e $\neg$. Mostramos abaixo esse resultado. Na verdade, apenas as porta $\wedge$ e $\neg$ (ou $\vee$ e $\neg$) são suficientes, pois $\vee$ pode ser simulada com $\wedge$ e $\neg$ (e $\wedge$ por $\vee$ e $\neg$), mas não vamos nos preocupar com tais questões de minimalidade no momento.

Teorema 5.2. *O conjunto $S = \{\wedge, \vee, \neg\}$ é um conjunto universal.*

Demonstração. A construção de todas as funções com imagem em F_2^m dá-se através da construção de uma seqüência ordenada de m circuitos que calculam funções booleanas com imagem em F_2, onde a saída de cada circuito corresponde a uma das m saídas da função original. Logo, podemos restringir nossa atenção às funções $F_2^n \to F_2^m$ para $m = 1$.

Para verificar que o conjunto $S = \{\wedge, \vee, \neg\}$ é universal, devemos mostrar que para toda função $f : F_2^n \to F_2$ existe um circuito booleano composto por portas de S com n entradas e uma saída que devolve as mesmas respostas que f para todas as possíveis entradas em F_2^n.

Sabemos que existe um conjunto $D \subseteq F_2^n$ tal que $f(x) = 1$ se e somente se $x \in D$. Suponha que $a = (a_1, \ldots, a_n)$ é um elemento de D. Nesse caso, o circuito booleano que calcula f deve devolver 1 sempre que $x_i = a_i$ para todo i, $1 \leq i \leq n$.

Com essa observação, é fácil ver que podemos construir uma função booleana $M_a(x_1, \ldots, x_n) = \phi(x_1) \wedge \phi(x_2) \ldots \wedge \phi(x_n)$ onde $\phi(x_i) = x_i$ se $a_i = 1$ e $\phi(x_i) = \neg x_i$ se $a_i = 0$. É claro que $M_a(x_1, \ldots, x_n) = 1$ se e somente se a entrada recebida for $(a_1, \ldots, a_n)$. Como podemos construir uma função com essa característica para todos os elementos de D, basta construir um circuito que calcula a função $f = M_{y_1} \vee M_{y_2} \vee \ldots \vee M_{y_k}$, onde $\{y_1, y_2, \ldots, y_k\} = D$. Com isso, temos que por meio de circuitos booleanos construídos com as portas de S podemos calcular todas as funções $f : F_2^n \to F_2$ para n fixo. Logo, S é um conjunto universal. □

Denote por F_2^* o conjunto $\bigcup_{n \geq 1} F_2^n$. Seja $f : F_2^* \to F_2$. Denote por f_n

a restrição de f ao domínio F_2^n. Já vimos que cada f_n tem um circuito C_n associado. Dizemos que $C_0, C_1, C_2, \ldots$ é uma *família de circuitos booleanos* que computa f. Podemos utilizar f para definir uma linguagem L_f em $\{0,1\}^*$, onde $x \in L_f$ se $f(x) = 1$ e $x \notin L_f$ se $f(x) = 0$. Neste caso, dizemos que $C_0, C_1, \ldots$ *decide* L_f. Se para uma entrada x o circuito $C_{|x|}$ devolve 1 como resposta, dizemos que $C_{|x|}$ *aceitou* x, e se $C_{|x|}$ devolve 0, dizemos que ele *rejeitou* x.

Se existe uma MT que, com entrada 1^n, produz C_n como saída (o grafo dirigido acíclico associado) para todo n em espaço $O(\lg n)$ (desconsiderando o espaço da entrada fornecida), então dizemos que $C_0, C_1, C_2, \ldots$ é uma *família uniformemente polinomial de circuitos* que decide L_f.

O seguinte resultado, demonstrado no livro de Papadimitriou [Pap94], mostra a importância da classe composta por linguagens que admitem famílias uniformemente polinomiais de circuitos.

Teorema 5.3. *Uma linguagem L está em* **P** *se e somente se pode ser decidida por uma família uniformemente polinomial de circuitos.* □

5.5.2 Circuitos reversíveis

Para estabelecer a ligação entre circuitos clássicos e quânticos, discutimos agora os circuitos reversíveis. Na mecânica quântica, todas as transformações envolvidas são unitárias (e portanto, reversíveis). Uma *transformação U é unitária* se $U^* = U^{-1}$, onde U^* é a transposta conjugada de U e U^{-1} é a inversa de U.

Os circuitos reversíveis são compostos por portas lógicas reversíveis. Uma porta lógica *reversível* em k bits é uma função inversível $F_2^k \to F_2^k$ para alguma constante k. Informalmente, a idéia é a seguinte: a partir da saída produzida pela porta, devemos ser capazes de determinar a entrada que foi utilizada.

Um exemplo de porta reversível é a chamada porta de *Toffoli*. Um esquema da porta segue abaixo. A partir de uma entrada $(x_1, x_2, x_3) \in F_2^3$, essa porta produz como saída $(y_1, y_2, y_3) \in F_2^3$, onde $y_1 = x_1$, $y_2 = x_2$ e $y_3 = \neg x_3$ se $x_1 = x_2 = 1$ e $y_3 = x_3$ caso contrário. É fácil ver que tal porta é reversível. Em particular, ela é a inversa dela mesma. Denotamos o uso da porta de Toffoli em uma entrada (x_1, x_2, x_3) por $T(x_1, x_2, x_3)$.

Outro exemplo de porta reversível, muito parecida com a porta de Toffoli, é a chamada porta *controlled not*. Um esquema da porta segue abaixo. A porta recebe um elemento (x_1, x_2) de F_2^2 e produz como saída um elemento (y_1, y_2) de F_2^2, onde $y_1 = x_1$ e $y_2 = x_2$ se $x_1 = 0$ e $y_1 = x_1$ e $y_2 = \neg x_2$ se $x_1 = 1$. Denotamos o uso da porta controlled not em uma entrada (x_1, x_2) por $C(x_1, x_2)$.

Um *circuito reversível* é um circuito booleano composto apenas por portas lógicas reversíveis que computa uma permutação em F_2^n, onde n é o tamanho

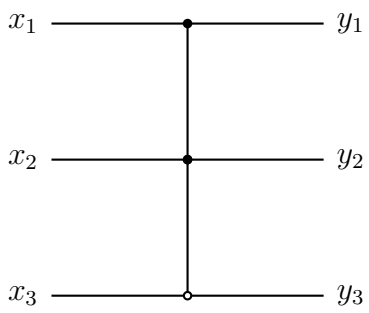


Figura 5.1: Porta de Toffoli

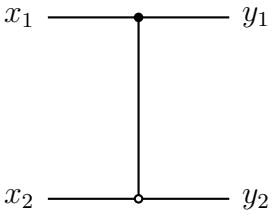


Figura 5.2: Porta controlled not

da palavra de entrada. Aplica-se aos circuitos reversíveis o conceito de conjunto universal de portas lógicas de maneira análoga a que vimos anteriormente. Se uma porta lógica reversível compõe sozinha um conjunto universal, dizemos que essa é uma *porta universal*. Fixado um conjunto universal, a complexidade de um circuito reversível C é o número de portas (reversíveis) que compõem C.

Um fato importante relacionado às portas reversíveis envolve a quantidade de bits que aparecem como entrada e como saída. Analisando a porta de Toffoli, por exemplo, vemos que os dois primeiros bits da saída são sempre iguais aos dois primeiros bits da entrada. A presença de tais bits, porém, é essencial na determinação da entrada a partir da saída, e portanto eles não podem ser descartados durante uma computação reversível. Tal fato não ocorre somente com portas reversíveis, mas também com circuitos reversíveis. Logo, quando consideramos computações reversíveis, é normal levar em conta o uso de bits "extras", utilizados para manter a reversibilidade das operações.

Assim, ao computar uma função $F_2^m \to F_2^m$, em geral construímos circuitos que calculam funções $F_2^{m+n} \to F_2^{m+n}$ para um n adequado, onde os m primeiros bits de entrada compõem a entrada propriamente dita (um elemento de F_2^m) e os n últimos são inicializados com algum elemento fixo de F_2^n. Podemos assumir que os m primeiros qubits de saída compõem a resposta procurada (também um elemento de F_2^m), e os n últimos garantem a reversibilidade do circuito, sendo utilizados apenas se quisermos obter a entrada original.

Já vimos que para toda função $F_2^n \rightarrow F_2^m$ existe um circuito booleano capaz de computá-la. Considerando as observações acima, temos que o mesmo resultado vale para funções inversíveis quando substituímos circuitos booleanos por circuitos reversíveis.

Já vimos que o conjunto $\{\wedge, \vee, \neg\}$ é um conjunto universal. Para ver que todo circuito booleano pode ser simulado por um circuito reversível, observamos que:

- A porta $\neg$ é reversível;
- A função $\wedge(x_1, x_2)$ pode ser computada reversivelmente se utilizarmos a porta de Toffoli com entrada $(x_1, x_2, 0)$ em seu lugar. O valor da terceira coordenada de $T(x_1, x_2, 0)$ é equivalente a $\wedge(x_1, x_2)$;
- Como $\vee(x_1, x_2) = \neg(\wedge(\neg(x_1), \neg(x_2)))$, toda porta $\vee$ pode ser substituída por um conjunto de portas $\wedge$ e $\neg$, e portanto também pode ser calculada reversivelmente;
- É possível simular a produção de várias saídas idênticas pela mesma porta por meio de uma porta controlled not e de constantes 0. Basta notar que $C(x_1, 0) = (x_1, x_1)$.

Assim, utilizando portas de Toffoli, portas controlled not, portas $\neg$ e bits 0, é possível construir um circuito reversível capaz de simular qualquer circuito booleano. Se bits 1 também forem utilizados, não precisamos das portas controlled not e $\neg$. Basta utilizar portas de Toffoli e as constantes 0 e 1. Assim, temos que a porta de Toffoli é uma porta reversível universal.

Por fim, destacamos que para cada circuito booleano de complexidade polinomial existe um circuito reversível de complexidade polinomial. Basta notar que nas trocas sugeridas nas observações acima uma porta lógica sempre é substituída por uma quantidade constante de portas reversíveis. Assim, temos o seguinte resultado, baseado no teorema 5.3:

Teorema 5.4. *Uma linguagem L em* **P** *pode ser decidida por uma família uniformemente polinomial de circuitos reversíveis.* □

5.5.3 Circuitos quânticos

Os circuitos clássicos são compostos por portas lógicas que atuam sobre bits. Já os circuitos quânticos são compostos por portas quânticas que atuam sobre superposições.

Uma *porta quântica* é uma função bijetora definida num espaço $\mathcal{H}_{2^k}$ para alguma constante inteira positiva k. Seja $f : \mathcal{H}_{2^n} \rightarrow \mathcal{H}_{2^n}$ uma função bijetora,

seja $|\phi\rangle$ um elemento de $\mathcal{H}_{2^n}$ e $|\psi\rangle = f(|\phi\rangle)$. Definimos *circuitos quânticos* de maneira análoga a que definimos os circuitos booleanos. A diferença é que nesse caso as portas lógicas são substituídas por portas quânticas, e tanto os vértices da entrada como da saída contêm superposições. Ou seja, cada vértice de entrada e de saída contém um elemento de $\mathcal{H}_2$.

Tanto as portas como os circuitos quânticos atuam sobre registradores quânticos e estão associados a transformações unitárias. Como toda transformação reversível também é unitária, temos que toda porta reversível pode ser vista como uma porta quântica. Conseqüentemente, todo circuito booleano reversível (polinomial) pode ser visto como um circuito quântico (polinomial).

Nos circuitos clássicos consideramos que as portas lógicas são ligadas por fios, mantendo assim a analogia com os circuitos do mundo real. No caso dos circuitos quânticos, utilizaremos essa mesma analogia por uma questão de comodidade. Porém, destacamos que, diferente do caso clássico, ela não se aplica na prática. A comunicação entre duas portas quânticas se faz por meio de qubits compartilhados. Denominaremos os qubits compartilhados por todas as portas de um circuito quântico de *qubits de comunicação*, pois a comunicação entre duas portas do circuito sempre vai ser feita através desses qubits. Já os qubits "exclusivos" (qubits compartilhados por um subconjunto específico das portas de um circuito quântico) são denominados *qubits privados*. Podemos assumir que no início da computação de um circuito, todos os qubits privados encontram-se no estado $|0\rangle$.

Os conceitos de complexidade de circuito, família de circuitos e de família uniformemente polinomial de circuitos aplicam-se também aos circuitos quânticos, e são definidos de maneira análoga. Assim, utilizando o teorema 5.4, concluímos que toda linguagem em $\mathbf{P}$ pode ser decidida por uma família de circuitos quânticos uniformemente polinomiais.

Um exemplo importante de porta quântica que atua em apenas um qubit é a *porta de Hadamard*

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \tag{5.5}$$

que transforma o qubit $|0\rangle$ no qubit

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big). \tag{5.6}$$

e transforma o qubit $|1\rangle$ no qubit

$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle\big). \tag{5.7}$$

Destacamos que, após a aplicação da porta de Hadamard em um qubit que está num estado puro, uma medição leva à obtenção de um dos dois estados básicos com a mesma probabilidade. Além disso, destacamos que a porta de Hadamard realiza a operação inversa dela mesma. Ou seja, aplicar a matriz de Hadamard duas vezes sobre o mesmo qubit não altera a superposição original do mesmo.

A toda porta quântica definida num espaço $\mathcal{H}_{2^n}$ está associada uma matriz M de dimensão 2^n. Denotaremos a matriz transposta conjugada de M por $M^\dagger$. Se $M^\dagger$ também é a matriz inversa de M, então dizemos que M é uma *matriz* (ou *transformação*) *hermitiana*.

Um artigo de Barenco et al. [BBC$^+$95] mostra que os circuitos quânticos podem ser construídos a partir de um conjunto universal, assim como ocorria com os circuitos clássicos.

Teorema 5.5. *Todos os circuitos quânticos podem ser construídos utilizando apenas portas controlled not e portas quânticas unárias.* □

Por fim, a leitura das respostas produzidas por um circuito quântico se faz por meio de operações de medição dos qubits de saída no estado final produzido.

5.6 Exemplo de algoritmo quântico

Algoritmos quânticos são métodos para resolver problemas algorítmicos que utilizam ações realizáveis por dispositivos quânticos. A noção de algoritmo quântico se confunde com a noção de circuito quântico. A seguir vamos apresentar um problema concreto e descrever um algoritmo quântico para ele (ou seja, um circuito quântico).

Dizemos que uma função f é dada como uma *caixa preta* se só podemos obter informações acerca de f através de sua aplicação a elementos de seu domínio. O *problema de Deutsch* consiste no seguinte. Seja $f : \{0,1\} \longrightarrow \{0,1\}$ uma função dada como uma caixa preta. Determine se $f(0) = f(1)$ ou se $f(0) \neq f(1)$. Em outras palavras, determine se f é constante ou "balanceada".

Para se resolver o problema com certeza no modelo clássico, são necessárias duas aplicações de f: uma para a entrada 0 e outra para a entrada 1. Vamos mostrar um algoritmo quântico, devido a Cleve, Ekert, Macchiavello e Mosca [CEMM98], que resolve o problema no modelo quântico com uma única aplicação de f.

Seja U_f a transformação unitária de dimensão 4 que leva $|xy\rangle$ a $\big|x(y\oplus f(x))\big\rangle$, onde $\oplus$ representa o operador "ou exclusivo". No modelo quântico, U_f é a transformação unitária que utiliza a caixa preta para f. O circuito que resolve o

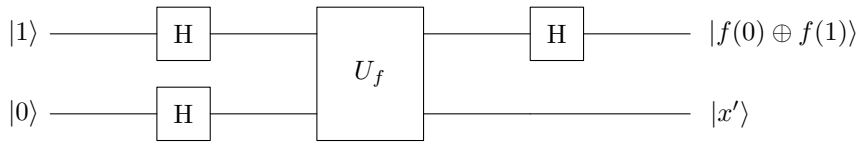


Figura 5.3: Circuito para o problema de Deutsch.

problema está descrito na figura acima. Vamos detalhar melhor o funcionamento do algoritmo.

Começamos com um registrador $|\phi_0\rangle$ de 2 qubits tal que $|\phi_0\rangle = |01\rangle$. Primeiro aplicamos a transformação de Hadamard nos 2 qubits do registrador, obtendo

$$\begin{aligned} |\phi_1\rangle &= (H \otimes H)|01\rangle \\ &= \frac{1}{2}\Big[|0\rangle \otimes \Big(|0\rangle - |1\rangle\Big)\Big] + \frac{1}{2}\Big[|1\rangle \otimes \Big(|0\rangle - |1\rangle\Big)\Big]. \end{aligned}$$

Neste ponto U_f é aplicada a $|\phi_1\rangle$ para obtermos $|\phi_2\rangle$:

$$\begin{aligned} |\phi_2\rangle &= U_f|\phi_1\rangle \\ &= \frac{1}{2}\Big\{U_f\Big[|0\rangle \otimes \Big(|0\rangle - |1\rangle\Big)\Big]\Big\} + \frac{1}{2}\Big\{U_f\Big[|1\rangle \otimes \Big(|0\rangle - |1\rangle\Big)\Big]\Big\}. \end{aligned}$$

Vamos escrever $|\phi_2\rangle$ de outra maneira. Para isso, vamos mostrar primeiro que

$$U_f\Big[|x\rangle \otimes \Big(|0\rangle - |1\rangle\Big)\Big] = (-1)^{f(x)}\Big[|x\rangle \otimes \Big(|0\rangle - |1\rangle\Big)\Big] \tag{5.8}$$

para $x \in \{0, 1\}$. De fato, temos que

$$\begin{aligned} |\psi\rangle &= U_f\Big[|x\rangle \otimes \Big(|0\rangle - |1\rangle\Big)\Big] = U_f\Big[|x0\rangle - |x1\rangle\Big] \\ &= \big|xf(x)\big\rangle - \big|x(1 \oplus f(x))\big\rangle. \end{aligned}$$

- Se $f(x) = 0$, então

$$|\psi\rangle = |x0\rangle - |x1\rangle = |x\rangle \otimes \Big(|0\rangle - |1\rangle\Big) = (-1)^{f(x)}\Big[|x\rangle \otimes \Big(|0\rangle - |1\rangle\Big)\Big].$$

- Se $f(x) = 1$, então

$$|\psi\rangle = |x1\rangle - |x0\rangle = |x\rangle \otimes \Big(|1\rangle - |0\rangle\Big) = (-1)^{f(x)}\Big[|x\rangle \otimes \Big(|0\rangle - |1\rangle\Big)\Big].$$

Provamos então a igualdade (5.8), de modo que,

$$\begin{aligned}
|\phi_2\rangle &= \frac{1}{2}\left\{(-1)^{f(0)}\Big[|0\rangle\otimes\Big(|0\rangle-|1\rangle\Big)\Big]\right\}+\frac{1}{2}\left\{(-1)^{f(1)}\Big[|1\rangle\otimes\Big(|0\rangle-|1\rangle\Big)\Big]\right\}\\
&= \frac{1}{2}\Big[\Big((-1)^{f(0)}|0\rangle+(-1)^{f(1)}|1\rangle\Big)\otimes\Big(|0\rangle-|1\rangle\Big)\Big]\\
&= \frac{1}{2}\Big[\Big((-1)^{f(0)}|0\rangle+(-1)^{f(1)}(-1)^{f(0)}(-1)^{f(0)}|1\rangle\Big)\otimes\Big(|0\rangle-|1\rangle\Big)\Big]\\
&= \frac{(-1)^{f(0)}}{2}\Big[\Big(|0\rangle+(-1)^{f(0)\oplus f(1)}|1\rangle\Big)\otimes\Big(|0\rangle-|1\rangle\Big)\Big]\\
&= (-1)^{f(0)}\left\{\left[\frac{1}{\sqrt{2}}\Big(|0\rangle+(-1)^{f(0)\oplus f(1)}|1\rangle\Big)\right]\otimes\left[\frac{1}{\sqrt{2}}\Big(|0\rangle-|1\rangle\Big)\right]\right\}.
\end{aligned}$$

Agora podemos aplicar a transformação de Hadamard ao primeiro qubit de $|\phi_2\rangle$ para obter

$$|\phi_3\rangle = (-1)^{f(0)}\left\{\Big[\big|f(0)\oplus f(1)\big\rangle\Big]\otimes\left[\frac{1}{\sqrt{2}}\Big(|0\rangle-|1\rangle\Big)\right]\right\}.$$

Uma medição do primeiro qubit de $|\phi_3\rangle$ fornece agora o valor de $f(0)\oplus f(1)$ e portanto o algoritmo descobre com certeza se f é constante ou "balanceada" através de uma única aplicação de f.

Esse exemplo mostra como se dá a descrição de um algoritmo quântico enquanto demonstra também como o modelo quântico de computação pode ser usado para resolver mais eficientemente um problema simples.

5.7 Classes quânticas de complexidade

Vamos introduzir agora duas classes quânticas de complexidade, tendo como base o modelo de circuitos. Essas classes foram propostas por Bernstein e Vazirani [BV97].

Vimos que a decisão de uma linguagem L em **P** pode ser feita por uma família uniformemente polinomial de circuitos. As definições das classes de linguagens em função de circuitos são feitas por meio de restrições das famílias de circuitos que decidem as linguagens das classes em questão.

Dizemos que uma linguagem L pertence à classe **EQP** (*exact quantum polynomial-time*) se existe uma família uniformemente polinomial de circuitos quânticos $C_0, C_1, \ldots$ tal que para toda entrada x em $\{0,1\}^*$ o circuito $C_{|x|}$

aceita x se e somente se $x \in L$. Essa classe corresponde à classe **P** do modelo clássico.

Dizemos que uma linguagem L pertence à classe **BQP** (*bounded error quantum polynomial-time*) se existe uma família uniformemente polinomial de circuitos quânticos $C_0, C_1, \ldots$ tal que para toda entrada x em $\{0,1\}^*$

1. Se $x \in L$ então $\Pr\left[C_{|x|} \text{ aceitar } x\right] \geq 2/3$;
2. Se $x \notin L$ então $\Pr\left[C_{|x|} \text{ rejeitar } x\right] \geq 2/3$.

A classe correspondente a **BQP** no modelo clássico é **BPP**, e assim como no caso desta, a probabilidade exigida de a resposta devolvida ser correta pode ser qualquer valor constante no intervalo (0,5; 1).

A seguir, são apresentadas relações envolvendo essas duas classes e suas correspondentes no modelo clássico. Esses dois resultados foram apresentados por Bernstein e Vazirani [BV97].

Teorema 5.6. $\mathbf{P} \subseteq \mathbf{EQP}$.

Demonstração. Seja L uma linguagem que pertence à classe **P**. Pelo teorema 5.4, sabemos que L pode ser decidida por uma família uniformemente polinomial $\mathcal{C}$ de circuitos reversíveis. Como toda porta reversível é uma porta quântica, temos que $\mathcal{C}$ corresponde a uma família uniformemente polinomial de circuitos quânticos que aceitam qualquer entrada x se e somente se $x \in L$. Logo, pela definição de **EQP**, concluímos que $L \in \mathbf{EQP}$, e assim concluímos que $\mathbf{P} \subseteq \mathbf{EQP}$. □

Aqui, apenas formalizamos esse resultado, que podia ser facilmente inferido a partir da afirmação final da seção 5.5.3. Mostramos agora um resultado mais interessante, que deixa explícita uma característica muito importante do modelo quântico de computação.

Teorema 5.7. $\mathbf{BPP} \subseteq \mathbf{BQP}$.

Demonstração. Seja L uma linguagem que pertence à classe **BPP**. Pela definição apresentada no capítulo 2, a linguagem L pode ser decidida por uma máquina de Turing probabilística M em tempo polinomial. Seja $p(|x|)$ o polinômio que limita o tempo consumido por M para uma entrada x.

Para decidir L para uma entrada x, M utiliza uma seqüência de bits aleatórios de comprimento no máximo $p(|x|)$ (polinomial em $|x|$). Gerar tal seqüência é trivial no modelo quântico. Vimos que, ao aplicar a porta de Hadamard num qubit que está num estado básico e realizar em seguida uma medição nesse qubit, obtemos 0 e 1 com probabilidade 1/2 para cada.

Logo, como são necessários $p(|x|)$ bits aleatórios na computação, basta ao circuito utilizar $p(|x|)$ transformações de Hadamard e $p(|x|)$ medições nos qubits que guardarão os elementos aleatórios. Ao final dessas operações, uma seqüência de $p(|x|)$ bits aleatórios terá sido gerada em $2p(|x|)$ passos, consumindo tempo polinomial em $|x|$.

O comportamento de M com uma entrada x pode ser simulado por uma máquina de Turing determinística N se fornecermos como entrada para N a seqüência de bits aleatórios utilizada por M (junto com a própria entrada x). Tal simulação consome tempo limitado por $p(|x|)$. Dessa forma, temos pelo teorema 5.3 que existe uma família uniformemente polinomial de circuitos que, utilizando uma seqüência de bits aleatórios que recebem como entrada, decide L.

A partir dos comentários feitos no final da seção 5.5.3, concluímos que existe uma família $\mathcal{C} = \{C_0, C_1, C_2, \ldots\}$ de circuitos quânticos uniformemente polinomiais tal que

1. Se $x \in L$ então $\Pr\left[C_{|x|} \text{ aceitar } x\right] \geq 2/3$;
2. Se $x \notin L$ então $\Pr\left[C_{|x|} \text{ rejeitar } x\right] \geq 2/3$.

Conforme observado, a seqüência de bits aleatórios não precisa ser fornecida como entrada. Cada circuito de $\mathcal{C}$ é capaz de gerar os bits aleatórios que utilizará em sua computação. Logo, pela definição de **BQP**, concluímos que $L \in \mathbf{BQP}$, e portanto $\mathbf{BPP} \subseteq \mathbf{BQP}$. □

É importante destacar que a prova de que **BPP** está contida em **BQP** mostra uma característica muito importante do modelo quântico de computação: a presença inerente de aleatoriedade pura. Utilizando transformações de Hadamard e medições, os circuitos quânticos são capazes de gerar seqüências de bits aleatórios. Não se conhecem métodos tão simples no modelo clássico de geração de seqüências de bits aleatórios.

Capítulo 6

Sistemas Interativos Quânticos

Nesse capítulo, apresentamos os sistemas interativos quânticos de prova e um importante resultado envolvendo tais sistemas.

6.1 Elementos e definições

No capítulo 4, vimos os sistemas interativos de prova, compostos por duas máquinas de Turing que interagiam entre si, sendo uma probabilística polinomialmente limitada e a outra infinitamente poderosa.

Em 2003, Watrous [Wat03] definiu o equivalente quântico desses sistemas, os chamados *sistemas interativos quânticos de prova* (**SIQP**), que apresentamos a seguir utilizando circuitos quânticos.

Definição

Um **SIQP** de m mensagens para uma palavra x em F_2^* é composto por um verificador de m mensagens e um provador de m mensagens. Seja $\mathcal{H}_{2^*} = \bigcup_n \mathcal{H}_{2^n}$. Um *verificador de m mensagens* é uma seqüência V de $k = \lfloor m/2 + 1 \rfloor$ transformações unitárias em $\mathcal{H}_{2^*}$. Cada uma dessas transformações será denotada por $V(x, j)$, onde j é a posição da transformação na seqüência V. Todas as transformações de V são computáveis por circuitos quânticos de complexidade polinomial.

Um *provador de m mensagens* é uma seqüência P de $k = \lfloor m/2+1/2 \rfloor$ transformações unitárias em $\mathcal{H}_{2^*}$. Cada uma dessas transformações será denotada por $P(x, j)$, onde j é a posição da transformação na seqüência de P. Todas as transformações de P são computáveis por circuitos quânticos cuja complexidade não é necessariamente polinomial.

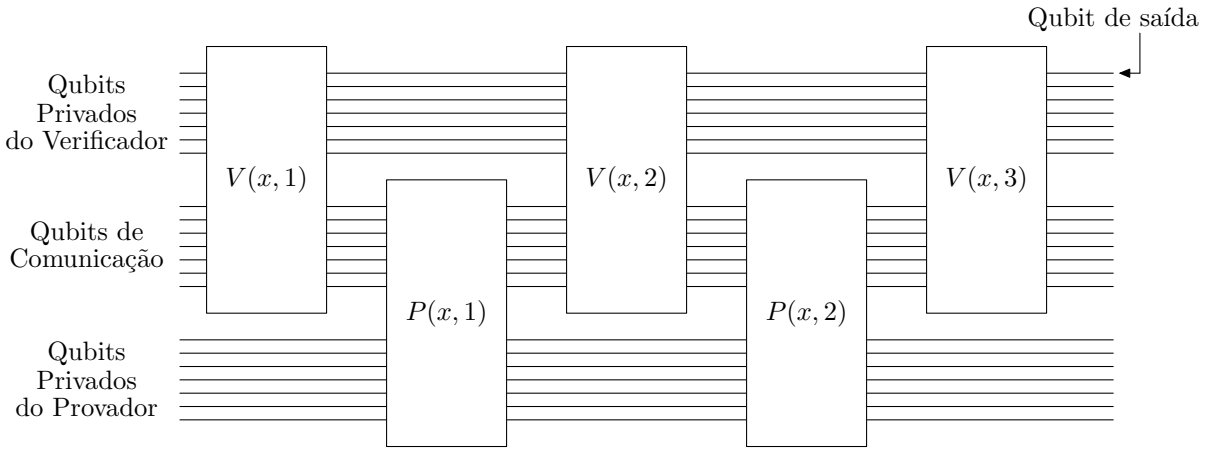


Figura 6.1: Circuito quântico para um sistema interativo quântico de 4 mensagens.

Dado um par (P, V), montamos um circuito quântico no qual os circuitos do provador e do verificador se alternam. O esquema de um circuito para um **SIQP** de 4 mensagens aparece na figura 6.1. Os qubits que passam diretamente de $V(x, i)$ para $V(x, i+1)$ (para todo i) são os qubits privados do verificador, enquanto que os que passam diretamente de $P(x, i)$ para $P(x, i+1)$ (para todo i) são os qubits privados do provador. Os demais qubits são de comunicação, e correspondem às mensagens trocadas entre o verificador e o provador.

A construção é intuitiva quando comparada com o funcionamento dos **SIP**s. Em função da entrada x, o provador (verificador) realiza suas computações, envia uma mensagem para o verificador (provador) por meio dos qubits de comunicação e aguarda até que o verificador (provador) envie uma mensagem (também por meio dos qubits de comunicação). O processo se repete até o término da interação. O último a receber uma mensagem é o verificador, que realiza suas computações finais e decide se aceita ou não a entrada x por meio da medição de um de seus qubits privados, o *qubit de saída*. Se a medição resultar em 1 ao final da computação, V aceita x. Caso contrário, V rejeita x.

Dizemos que uma linguagem L admite um *sistema interativo quântico de prova* de m mensagens com probabilidade de erro ϵ se, para cada x em Σ^*, existe um verificador V de m mensagens tal que:

1. Existe um provador P de m mensagens tal que se $x \in L$ então (P, V) sempre aceita x.

2. Para todo provador P' de m mensagens, se $x \notin L$ então (P', V) aceita x com probabilidade limitada superiormente por ϵ.

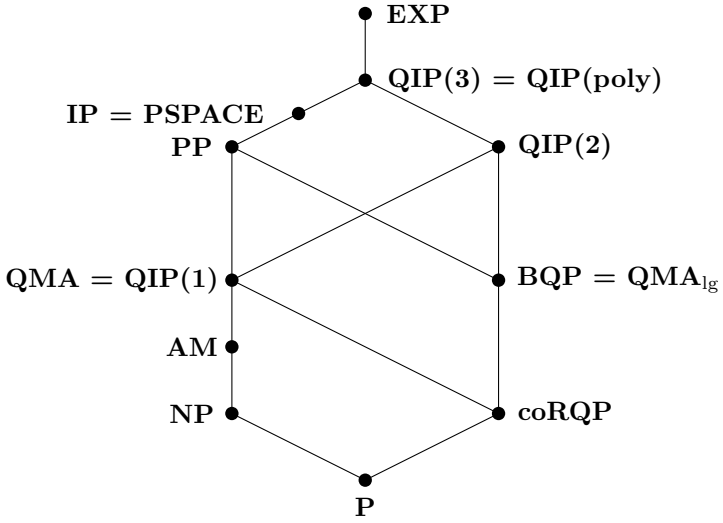
Denotamos por $\mathbf{QIP}(t(n))$ a classe das linguagens que admitem sistemas interativos quânticos de prova em que o número de trocas de mensagens é limitado superiormente por $t(n)$, onde n é o comprimento da entrada.

As linguagens que admitem sistemas interativos quânticos de prova de uma mensagem compõem a classe **QMA** (ou seja, $\mathbf{QMA} = \mathbf{QIP}(1)$). Essa classe foi definida originalmente por Watrous [Wat02], e alguns resultados a respeito dela aparecem no artigo de Marriott e Watrous [MW04].

Por fim, a classe das linguagens que admitem sistemas interativos quânticos de prova de uma mensagem nos quais o comprimento da única mensagem do provador para o verificador é $O(\lg n)$, onde n é o comprimento da palavra dada, é denotada por $\mathbf{QMA}_{\lg}$. Resultados envolvendo essa classe são também apresentados neste mesmo artigo de Marriott e Watrous [MW04].

Relações

O número de trocas de mensagens é extremamente importante nos sistemas interativos quânticos de prova. Abaixo, mostramos uma figura com as relações conhecidas entre as classes $\mathbf{QIP}(1)$, $\mathbf{QIP}(2)$ e $\mathbf{QIP}(3)$ e algumas classes de complexidade clássicas e quânticas.



A classe **PP** é a classe das linguagens decidíveis em tempo polinomial por uma máquina de Turing não-determinística onde se a palavra está na linguagem então mais da metade das possíveis computações a aceitam, e se a palavra não está, pelo mais da metade das possíveis computações a rejeitam. A classe **EXP** é a classe das linguagens decididas por máquinas de Turing determínisticas que consomem tempo limitado exponencialmente no tamanho da entrada.

A classe **AM** é definida por $\mathbf{AM} = \bigcup_{k \geq 0} \mathbf{AM}(k)$. Ou seja, é a união das classes **AM** em que o número de trocas de mensagens nos protocolos é constante. Já a classe $\mathbf{QIP}(poly)$ é a classe composta por linguagens decididas por sistemas interativos quânticos que utilizam um número polinomial de trocas de mensagens.

Por fim, a classe **RP** é a classe das linguagens decidíveis em tempo polinomial por uma máquina de Turing probabilística tal que se a palavra dada não está na linguagem, a máquina sempre a rejeita, e se ela está na linguagem, é aceita por pelo menos metade das computações. A classe **RQP** é a versão quântica da classe **RP**, e **coRQP** é seu complemento.

6.2 PSPACE $\subseteq$ QIP(3)

Ao estudar os sistemas interativos de prova no modelo clássico, vimos que **PSPACE** = **IP**. Na demonstração apresentada no capítulo 4, o número de mensagens trocadas entre o provador e o verificador é polinomial no tamanho da entrada. Não se conhece uma prova desse resultado com um protocolo no qual o número de trocas de mensagens seja constante. Veremos agora um resultado envolvendo os **SIQP**s, que mostra que a classe **PSPACE** está contida na classe **QIP**(3).

Na demonstração do resultado **PSPACE** = **IP**, mostramos que o problema QBF, que é **PSPACE**-completo, admite uma prova interativa com um número polinomial de trocas de mensagens. Aqui, utilizaremos o mesmo problema, e vamos mostrar que ele pode ser decidido com probabilidade de erro exponencialmente pequena por um sistema interativo quântico de prova que realiza três trocas de mensagens.

Teorema 6.1. PSPACE $\subseteq$ QIP(3).

Demonstração. O protocolo apresentado na demonstração do teorema 4.3 será essencial nessa prova. Quando estivermos nos referindo a algum elemento do protocolo clássico (como o provador, o verificador e as mensagens por eles enviadas), utilizaremos os termos *provador clássico*, *verificador clássico* e assim por diante. Assim como na demonstração do teorema 4.3, vamos denotar por $\phi = Q_1 x_1 \ldots Q_n x_n B(x_1, \ldots, x_n)$ a instância de QBF de tamanho n que está sendo utilizada como entrada para o protocolo.

Para mostrar que **PSPACE** $\subseteq$ **QIP**(3), vamos descrever um sistema interativo quântico de prova de três mensagens que decide o problema QBF com probabilidade de erro exponencialmente pequena em função do tamanho da instância.

Na demonstração do teorema 4.3, as cotas superiores para a probabilidade de erro eram obtidas por meio de execuções independentes do mesmo protocolo. Porém, não levamos em conta esse fato na descrição do protocolo. No caso quântico, execuções múltiplas também são importantes para garantir a limitação da probabilidade de erro. Vamos apresentar um protocolo em que as múltiplas execuções são simultâneas. Na descrição, em vez de considerar apenas uma computação, temos que considerar todas simultaneamente para discutir o grau de independência entre elas.

O número de execuções simultâneas será denotado por m. O valor de m é polinomial em n, mas deixaremos para fixá-lo posteriormente. A escolha apropriada de m é crucial para a obtenção de um protocolo com probabilidade de erro exponencialmente baixa.

Elementos da prova

O conjunto de qubits de comunicação contém a descrição de três matrizes, que denotaremos por $\mathbf{R}$, $\mathbf{S}$ e $\mathbf{P}$. Quando estivermos nos referindo a uma possível configuração básica dessas matrizes (e não à superposição de configurações que elas de fato contêm), utilizaremos como notação R, S e P, respectivamente.

Essas matrizes têm m linhas e N colunas, onde $N = \frac{n^2+3n}{2}$ é o número de polinômios enviados pelo provador clássico e o número de valores aleatórios do corpo F sorteados pelo verificador clássico na prova do teorema 4.3 (lembre-se que n é o tamanho de ϕ). Cada posição dessas matrizes é um registrador, que será representado por $\mathbf{R}_{i,j}$, $\mathbf{S}_{i,j}$ ou $\mathbf{P}_{i,j}$ para cada $1 \leq i \leq m$ e $1 \leq j \leq N$.

Nessa prova, o tamanho do corpo F utilizado no protocolo clássico é importante. Porém, ao invés de escolher para o tamanho de F um valor polinomial em n como fizemos na prova do teorema 4.3, escolheremos um valor exponencial em n. Mais precisamente, assumimos que $|F| = 2^k$, onde o valor de k é polinomial em n. A escolha de k é importante para a limitação da probabilidade de erro do protocolo, e será feita mais tarde.

As matrizes $\mathbf{R}$ e $\mathbf{S}$ são utilizadas para armazenar os elementos de F utilizados pelo verificador no protocolo clássico. Cada registrador $\mathbf{R}_{i,j}$ e $\mathbf{S}_{i,j}$ contém k qubits, podendo assim armazenar qualquer um dos 2^k elementos de F. Cada linha dessas matrizes vai guardar N valores de F que podem ser utilizados como os elementos sorteados em uma interação clássica pelo verificador. Mais do que isso, cada registrador não vai conter apenas um estado básico, mas sim uma superposição. Logo, em cada uma das m linhas dessas matrizes há uma superposição de seqüências de valores "aleatórios".

Já os registradores de $\mathbf{P}$ são utilizados para guardar os polinômios usados pelo provador clássico em interações do protocolo clássico. Cada uma das m

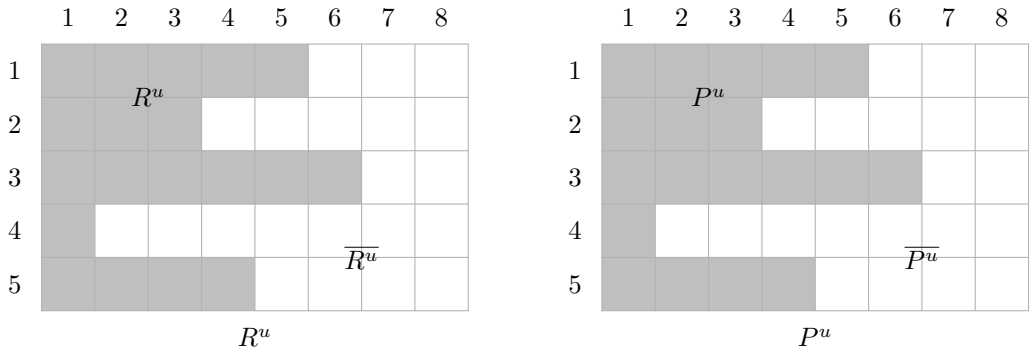


Figura 6.2: Exemplo de divisão de $\mathbf{R}$ e $\mathbf{P}$ para $N = 8$, $m = 5$ e $u = (6, 4, 7, 2, 5)$.

linhas da matriz $\mathbf{P}$ contém uma superposição de seqüências de N polinômios. Como os polinômios utilizados pelo provador clássico têm coeficientes em F e devem ter grau máximo n, cada registrador $\mathbf{P}_{i,j}$ consiste em $n + 1$ conjuntos de k qubits, onde cada conjunto representa um coeficiente de um polinômio.

Por fim, ainda dentro do conjunto de qubits de comunicação, encontramos o vetor $u \in \{1, \ldots, N\}^m$, ou seja, um vetor com m registradores. Ele será utilizado pelo verificador do protocolo quântico para guardar uma seqüência de valores gerados aleatoriamente.

Denotaremos por $\mathbf{R}_i$ o conjunto de registradores $\mathbf{R}_{i,1}, \mathbf{R}_{i,2}, \ldots, \mathbf{R}_{i,N}$ (ou seja, a i-ésima linha de R) e por $\mathbf{R}^u$ o conjunto de registradores tal que, para cada i, se a i-ésima posição de u é u_i, então $\mathbf{R}^u$ contém os elementos $\mathbf{R}_{i,1}, \ldots, \mathbf{R}_{i,u_i-1}$. Por fim, $\overline{\mathbf{R}^u}$ é o "complemento" de $\mathbf{R}^u$ em relação a $\mathbf{R}$, ou seja, para cada i, $\overline{\mathbf{R}^u}$ contém os elementos $\mathbf{R}_{i,u_i}, \mathbf{R}_{i,u_{i+1}}, \ldots, \mathbf{R}_{i,N}$. Veja um exemplo na figura 6.2. Essas notações serão utilizadas de maneira análoga para $\mathbf{S}$ e $\mathbf{P}$.

O protocolo e a aceitação de instâncias verdadeiras

Vamos agora descrever em detalhes o funcionamento do protocolo, mostrando que se ϕ é verdadeira, então existe um provador que faz o verificador aceitar a prova sempre. Na figura 6.3, apresentamos os algoritmos do provador e do verificador para mostrar o funcionamento do protocolo.

Inicialmente, o provador aplica a transformação de Hadamard em todos os qubits de $\mathbf{R}$. Como cada posição de $\mathbf{R}$ possui k qubits, cada $\mathbf{R}_{i,j}$, $1 \leq i \leq m$ e $1 \leq j \leq N$, vai conter uma superposição onde os elementos de F são obtidos após uma medição com a mesma probabilidade. Em resumo, um estado básico R da superposição de $\mathbf{R}$ pode ser visto como uma matriz contendo valores aleatórios de F. Todos os possíveis valores de R estão presentes em $\mathbf{R}$.

Em seguida, o provador faz uma cópia de $\mathbf{R}$ em $\mathbf{S}$, de modo que as duas

matrizes sejam idênticas e estejam entrelaçadas (ou seja, se modificamos um qubit em $\mathbf{R}$, esse será modificado em $\mathbf{S}$ também e vice-versa).

Depois, a partir da matriz $\mathbf{R}$, o provador gera uma matriz $\mathbf{P}$ formada por estados básicos $P(R)$ (quando o R ao qual P está associado for claro, vamos denotar $P(R)$ apenas por P), contendo os polinômios que um provador utilizará no protocolo clássico caso o verificador clássico utilize R como fonte de seqüências aleatórias de F. Mais precisamente, para cada i, $P_{i,1}, P_{i,2}, \ldots, P_{i,N}$ é a seqüência de polinômios enviados por um provador no protocolo clássico quando os valores aleatórios utilizados pelo verificador clássico são $R_{i,1}, \ldots, R_{i,N}$. A geração é feita por meio da operação T, usada na linha 3 do algoritmo do provador. Essa operação nada mais é do que a implementação quântica do protocolo clássico. Para cada estado básico R em $\mathbf{R}$, é gerado um estado básico $P(R)$ em $\mathbf{P}$.

Um provador que gera as matrizes $\mathbf{R}$, $\mathbf{S}$ e $\mathbf{P}$ da maneira descrita acima será chamado daqui para a frente de *provador honesto*.

Após a geração dessas matrizes, o estado do sistema é

$$2^{-kmN/2} \sum_{R} \{|R\rangle|R\rangle|P(R)\rangle\}_{\text{provador}}. \tag{6.1}$$

Com essa superposição em mãos, o provador envia uma mensagem para o verificador contendo $\mathbf{R}$ e $\mathbf{P}$, guardando para si a matriz $\mathbf{S}$. Sabemos que o verificador tem acesso à matriz $\mathbf{S}$ também, mas podemos assumir que ele não vai modificá-la, pois isso comprometeria o funcionamento do protocolo.

Após essa mensagem, o estado do sistema é

$$2^{-kmN/2} \sum_{R} \{|R\rangle|P(R)\rangle\}_{\text{verificador}} \{|R\rangle\}_{\text{provador}}. \tag{6.2}$$

Após receber a primeira mensagem do provador, o verificador faz sua primeira checagem. A prova será rejeitada se $(\mathbf{R}_i, \mathbf{P}_i)$, para qualquer linha $i \in \{1, \ldots, m\}$, contém uma prova inválida de que ϕ é verdadeira. A validade da prova está relacionada com o protocolo clássico. Ou seja, o verificador simula (reversivelmente) o protocolo clássico, utilizando os polinômios dados em $\mathbf{P}_i$ e os valores de F dados em $\mathbf{R}_i$ como os valores aleatórios do protocolo clássico e checa se o valor obtido pela simulação, após uma medição, é 1.

Para isso, o verificador avalia reversivelmente a instância ϕ de QBF para cada par $(\mathbf{R}_i, \mathbf{P}_i)$, e após isso realiza a medição no qubit que contém a resposta. Tal medição resulta em 1 se ϕ é verdadeira. Caso contrário, a medição resulta em 0 com alta probabilidade. Após a medição, se o valor obtido for 0, a prova é rejeitada. Senão, as configurações $(R_i, P(R)_i)$ inválidas (que possuem resposta 0) foram eliminadas por conta da medição, e o verificador realiza

a computação inversa para retornar ao estado enviado pelo provador. Essas operações estão comprimidas nos passos 2, 3 e 4 do algoritmo do verificador. Se ϕ é verdadeira e o provador é honesto, a prova nunca é rejeitada após essa checagem.

Feita essa primeira checagem, o verificador gera o vetor $u \in \{1, \ldots, N\}^m$, escolhendo cada um de seus m elementos aleatoriamente de maneira uniforme. O verificador envia u e $\overline{\mathbf{P}^u}$ ao provador. Como u não está entrelaçado com as outras três matrizes descritas nos qubits de comunicação, não vamos incluí-lo na descrição do estado do sistema, que nesse ponto é

$$2^{-kmN/2} \sum_R \{|R\rangle|P(R)^u\rangle\}_{\text{verificador}} \{|R\rangle|\overline{P(R)^u}\rangle\}_{\text{provador}}. \tag{6.3}$$

Nesse ponto, o verificador está com $\mathbf{R}$ e $\mathbf{P}^u$. Já o provador tem $\mathbf{S}$ e $\overline{\mathbf{P}^u}$.

Antes de descrever o segundo passo do provador, vamos definir a função utilizada na sexta linha de seu algoritmo. Para cada par (i, j), $1 \leq i \leq m$ e $1 \leq j \leq N$, $T_{i,j}$ representa a transformação unitária $T_{i,j} : |\mathbf{R}\rangle|0\rangle \rightarrow |\mathbf{R}\rangle|\mathbf{P}_{i,j}\rangle$.

A partir de u e $\overline{\mathbf{P}^u}$, o provador honesto utiliza $T_{i,j}^{-1}$ para levar todos os qubits dos registradores de $\overline{\mathbf{P}^u}$ para o estado $|0\rangle$. Após isso, o provador envia $\mathbf{S}$ para o verificador, que nesse ponto já não está entrelaçado com $\overline{\mathbf{P}^u}$. O estado do sistema passa a ser

$$2^{-kmN/2} \sum_R \{|R\rangle|R\rangle|P(R)^u\rangle\}_{\text{verificador}} \{|0\rangle\}_{\text{provador}}. \tag{6.4}$$

Deixamos claro que o $|0\rangle$ na descrição do estado do provador indica que todas as posições de $\overline{\mathbf{P}^u}$ estão no estado básico $|0\rangle$.

Em seu último passo, o verificador checa se cada qubit de $\mathbf{R}$ foi gerado de modo que $|0\rangle$ e $|1\rangle$ são obtidos após uma medição com a mesma probabilidade, se $\mathbf{R}$ e $\mathbf{S}$ são iguais e se os polinômios gerados pelo provador de fato são independentes das entradas de $\mathbf{R}$ correspondentes a elementos aleatórios utilizados em passos posteriores da interação clássica. Ou seja, para alguns polinômios $\mathbf{P}(R)_{i,j}$, o verificador checa se eles são independentes dos valores $\mathbf{R}_{i,j+1}, \mathbf{R}_{i,j+2}, \ldots, \mathbf{R}_{i,N}$. Se não for esse o caso, o provador quântico gerou algum polinômio após observar um valor ao qual o provador clássico não teria acesso. Isso é considerado "trapaça" e leva à rejeição da prova com alta probabilidade, como veremos.

Para fazer essa checagem, inicialmente o verificador subtrai $\mathbf{R}_{i,j}$ de $\mathbf{S}_{i,j}$ para cada i, j, $1 \leq i \leq m$ e $1 \leq j \leq N$. Se $\mathbf{R}$ e $\mathbf{S}$ foram gerados por um provador honesto, eles são iguais e portanto $\mathbf{S}$ é apagado (todos os seus qubits ficam iguais a $|0\rangle$). Em seguida, o verificador aplica a transformação de Hadamard nos

Algoritmo do Provador

1ª Fase

1 $\mathbf{R} \leftarrow \mathcal{H}_{2^{Nmk}}|0\rangle$

2 $\begin{pmatrix} \mathbf{R} \\ \mathbf{S} \end{pmatrix} \leftarrow \begin{pmatrix} I & 0 \\ I & I \end{pmatrix} \begin{pmatrix} \mathbf{R} \\ |0\rangle \end{pmatrix}$

3 $\begin{pmatrix} \mathbf{R} \\ \mathbf{P} \end{pmatrix} \leftarrow \begin{pmatrix} I & 0 \\ 0 & T \end{pmatrix} \begin{pmatrix} \mathbf{R} \\ |0\rangle \end{pmatrix}$

4 **Envia R** e **P** para o verificador

2ª Fase

5 **Recebe** u e $\overline{\mathbf{P}^u}$ do verificador

6 Aplica $T_{i,j}^{-1}$ em $\overline{\mathbf{P}^u}$ (e **S**)

7 **Envia S** para o verificador

Algoritmo do Verificador

1ª Fase

1 **Recebe R** e **P** do provador

2 Para i de 1 a m faça

3 Se (R_i, P_i) é inválido então

4 Rejeita a prova

5 Para i de 1 a m faça

6 $u_i \leftarrow \text{RAND}(N)$

7 **Envia** u e $\overline{\mathbf{P}^u}$ para o provador

2ª Fase

8 **Recebe S** do provador

9 $\mathbf{S} \leftarrow \mathbf{S} - \mathbf{R}$

10 Aplica Hadamard em cada qubit de $\overline{\mathbf{R}^u}$

11 Se algum qubit de algum registrador de $\overline{\mathbf{R}^u}$ é diferente de zero

12 Rejeita a prova

13 Aceita a prova

Figura 6.3: Algoritmos do provador e do verificador.

registradores de $\overline{\mathbf{R}^u}$. Se após isso todos os elementos de $\overline{\mathbf{R}^u}$ forem transformados em $|0\rangle$, a prova é aceita. Senão, ela é rejeitada.

Se o provador é honesto, $\overline{\mathbf{R}^u}$ não está entrelaçado com quaisquer outros registradores no momento da aplicação das transformações. Em especial, ele não estará entrelaçado com $\mathbf{P}^u$, o que mostra que os polinômios desses registradores são independentes dos elementos de F em $\overline{\mathbf{R}^u}$. Logo, se ϕ é verdadeira e o provador é honesto, o verificador sempre aceita a prova com esse protocolo.

Delimitação da probabilidade de erro

Vamos estudar agora a probabilidade de o verificador ser enganado nos casos em que ϕ não é verdadeira. Para isso, vamos analisar os estados do sistema conforme os passos do provador e do verificador vão sendo executados.

Antes disso, vamos analisar como o verificador pode ser enganado nesse protocolo. No caso do protocolo clássico, o provador dependia da sorte para conseguir convencer o verificador de que uma instância falsa do QBF era verdadeira. Porém, se o provador de alguma forma conhecesse pelo menos um valor aleatório que seria utilizado pelo verificador em uma mensagem posterior, ele poderia escolher um conjunto de polinômios a serem enviados que o auxiliariam a ludibriar o verificador.

No protocolo clássico isso não pode ocorrer porque a geração de valores aleatórios é feita pelo verificador, e ela só ocorre no momento em que esses valores vão ser enviados ao provador. No caso do protocolo quântico que descrevemos, quem gera os estados com as superposições de interações clássicas, incluindo os valores aleatórios, é o provador. Tendo acesso aos valores aleatórios, o provador pode tentar trapacear, escolhendo polinômios em função de valores que o provador clássico desconhece. Além disso, o provador pode não gerar de maneira honesta os valores aleatórios (fazendo com que alguns valores sejam gerados com probabilidade maior que outros, por exemplo).

A análise da probabilidade de decisão errada do verificador baseia-se nesses pontos. Vamos estudar a probabilidade de o verificador ser enganado quando o provador tenta fazer algum tipo de trapaça no protocolo.

Em seu primeiro passo, o provador envia $\mathbf{R}$ e $\mathbf{P}$ para o verificador. O estado do sistema nesse momento passa a ser

$$\sum_{R,P} \alpha(R,P)|R\rangle|P\rangle|\xi(R,P)\rangle. \tag{6.5}$$

Cada $\alpha(R,P)$ é um número complexo tal que $\sum_{R,P}|\alpha(R,P)|^2 = 1$ e $|\xi(R,P)\rangle$ é um vetor normalizado representando o estado dos qubits privados do provador

que eventualmente estão entrelaçados com $\mathbf{R}$ e $\mathbf{P}$. Destacamos que, ao contrário do que acontecia na seção anterior, aqui a somatória tem como índices R e P e não somente R, pois não podemos garantir que de fato $\mathbf{P}$ é totalmente determinado por $\mathbf{R}$ (só temos essa garantia quando o provador é honesto).

Como é do interesse do provador passar pelas checagens do verificador, podemos assumir que a superposição (6.5) contém apenas pares (R_i, P_i) válidos (se houver pares inválidos, ou a prova vai ser rejeitada ou esses pares sumirão após a checagem do verificador).

Depois, o verificador envia o vetor aleatório u e os registradores de $\overline{\mathbf{P}^u}$. O provador executa seu segundo passo e envia a matriz $\mathbf{S}$, da qual o verificador subtrai a matriz $\mathbf{R}$. Após essa operação, o estado do sistema passa a ser

$$\sum_{R,P^u} \beta(R,u,P^u)|R\rangle|P^u\rangle|\eta(R,u,P^u)\rangle, \tag{6.6}$$

onde novamente $\beta(R,P^u)$ é um número complexo tal que $\sum_{R,P^u}|\beta(R,u,P^u)|^2 = 1$ e $|\eta(R,u,P^u)\rangle$ é o estado normalizado que descreve os qubits privados do provador e a matriz $\mathbf{S}$ (já subtraída de $\mathbf{R}$), que agora está com o verificador.

Finalmente, o verificador aplica a transformação de Hadamard em todos os qubits de $\overline{\mathbf{R}^u}$ e rejeita a prova se alguma dessas medições não tem 0 como resultado. Denotando por $H^{\otimes k}$ a transformação unitária de Hadamard em k qubits (que é o tamanho de cada registrador em $\overline{\mathbf{R}^u}$), e denotando por l o número de registradores em $\overline{\mathbf{R}^u}$ (ou seja, $l = \sum_{i=1}^{m}(N-(u_i+1))$), temos que a probabilidade de aceitação da prova (ou seja, de o valor medido pelo verificador ser 1) é dada por:

$$\begin{aligned}\Big\| \sum_{R,P^u} \beta(R,u,P^u)|R^u\rangle\langle 0|H^{\otimes k}|R_{1,u_1}\rangle \\ \langle 0|H^{\otimes k}|R_{1,u_1+1}\rangle \ldots \langle 0|H^{\otimes k}|R_{m,N}\rangle|P^u\rangle|\eta(R,u,P^u)\rangle\Big\|^2.\end{aligned} \tag{6.7}$$

Ou seja, a cada par R, P^u está associado o estado puro $|R^u\rangle|P^u\rangle|\eta(R,u,P^u)\rangle$, e a amplitude de tal estado é dada pelo produto de $\beta(R,u,P^u)$ pelo número $\langle 0|H^{\otimes k}|R_{i,j}\rangle$.

Como cada registrador $\mathbf{R}_{i,j}$ é composto por k qubits, todo estado puro $|R_{i,j}\rangle$ está associado a um elemento de $\mathcal{H}_{2^k}$. A aplicação da transformação $H^{\otimes k}$ em qualquer elemento de $\mathcal{H}_{2^k}$ resulta na superposição $\sum_{x\in\{0,1\}^k} 2^{-k/2}|x\rangle$. Logo, para todo $R_{i,j}$, o número $\langle 0|H^{\otimes k}|R_{i,j}\rangle$ é igual a $2^{-k/2}$. Dessa forma, temos que

a expressão (6.7) é igual a:

$$\begin{aligned} & 2^{-lk} \left\| \sum_{R,P^u} \beta(R,u,P^u)|R^u\rangle|P^u\rangle|\eta(R,u,P^u)\rangle \right\|^2 \\ = \;& 2^{-lk} \sum_{R^u,P^u} \left\| \sum_{\overline{R^u}} \beta(R,u,P^u)|\eta(R,u,P^u)\rangle \right\|^2 \\ \leq \;& 2^{-lk} \sum_{R^u,P^u} \left(\sum_{\overline{R^u}} |\beta(R,u,P^u)| \right)^2 . \end{aligned} \tag{6.8}$$

A igualdade entre a primeira e a segunda linha vale porque os vetores $|R^u\rangle|P^u\rangle|\eta(R,u,P^u)\rangle$ são dois a dois ortogonais, e portanto a soma das normas é igual à norma da soma desses vetores. Já a segunda relação segue da desigualdade triangular.

Logo, basta limitar essa última expressão, assumindo que u é escolhido aleatoriamente de maneira uniforme (podemos assumir esse fato pois quem gera esse vetor é o verificador). Inicialmente, associamos a cada registrador $\mathbf{R}_{i,j}$ e $\mathbf{P}_{i,j}$, $1 \leq i \leq m$ e $1 \leq j \leq N$, uma variável aleatória que assume valores de F de acordo com a probabilidade de eles serem obtidos por meio de uma medição no estado (6.5) do sistema (no caso do provador honesto, a variável $\mathbf{R}_{i,j}$ segue a distribuição uniforme, e portanto a obtenção dos elementos de F após uma medição é equiprovável).

Inicialmente, afirmamos que, no estado (6.6)

$$|\beta(R,u,P^u)|^2 \;=\; \Pr\left[\mathbf{R} = R, \mathbf{P}^u = P^u\right]$$

para cada R e P^u possíveis, pois $\mathbf{R}$ e $\mathbf{P}^u$ não são manipulados nos passos executados entre o estado (6.5) e o estado (6.6). Assim, podemos reescrever (6.8) como

$$2^{-lk} \sum_{R^u,P^u} \left(\sum_{\overline{R^u}} \sqrt{\Pr[\mathbf{R} = R, \mathbf{P}^u = P^u]} \right)^2 . \tag{6.9}$$

Da definição de probabilidade condicional, temos o seguinte:

$$\begin{aligned} \Pr\left[\mathbf{R} = R, \mathbf{P}^u = P^u\right] \;&=\; \Pr\left[\overline{\mathbf{R}^u} = \overline{R^u}, \mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] \\ &= \; \Pr\left[\mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] \Pr\left[\overline{\mathbf{R}^u} = \overline{R^u} | \mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] . \end{aligned}$$

Para todo conjunto finito S e função $f : S \to [0,1]$, definimos $\theta_S(f) = \sum_{s \in S} \sqrt{f(s)}$. Para cada par R^u, P^u, vamos definir uma função $X_{R^u,P^u} : F^l \to$

$[0,1]$, onde l é o número de elementos de $\overline{\mathbf{R}^u}$, como sendo $X_{R^u,P^u}(\overline{R^u}) = \Pr\left[\overline{\mathbf{R}^u} = \overline{R^u} | \mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right]$.

Assim sendo, para cada R^u, P^u,

$$\begin{aligned}
&\left(\sum_{\overline{R}^u} \sqrt{\Pr[\mathbf{R} = R, \mathbf{P}^u = P^u]}\right)^2 \\
&= \Pr\left[\mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] \left(\sum_{\overline{R}^u} \sqrt{\Pr\left[\overline{\mathbf{R}^u} = R^u | \mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right]}\right)^2 \\
&= \Pr\left[\mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] \left(\theta_{F^l}(X_{R^u,P^u})\right)^2.
\end{aligned}$$

Portanto, podemos reescrever (6.9) como

$$2^{-lk} \sum_{R^u,P^u} \Pr\left[\mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] \left(\theta_{F^l}(X_{R^u,P^u})\right)^2. \tag{6.10}$$

Vamos definir agora eventos sobre as variáveis aleatórias (associadas a $\mathbf{R}_{i,j}$ e $\mathbf{P}_{i,j}$) que definimos. Para $1 \leq i \leq m$, $1 \leq j \leq N-1$ e uma matriz R, definimos

$A_{i,j}$: $\mathbf{P}_{i,j'}$ contém um polinômio distinto de $P_{i,j'}(R)$ para todo $j' \leq j$ mas $\mathbf{P}_{i,j+1}$ contém $P_{i,j+1}(R)$.

Intuitivamente, $A_{i,j}$ indica que o provador estava usando polinômios incorretos (i.e., polinômios diferentes dos que um provador honesto enviaria) na computação da linha i mas utilizou $P_{i,j+1}(R)$ na $(j+1)$-ésima coluna dessa linha.

Para $1 \leq i \leq m$, definimos

$A_{i,N}$: $\mathbf{P}_{i,j}$ contém um polinômio distinto de $P_{i,j}(R)$ para todo $j \leq N$.

Esse evento indica que, na computação descrita pela linha i, em nenhum momento o provador utiliza um polinômio correto.

Finalmente, definimos $B_u = \bigcup_{1 \leq i \leq m} A_{i,u_i}$. Esse evento indica que, tendo sorteado o vetor u, o verificador localiza pelo menos uma das posições a partir das quais o provador passou a utilizar polinômios corretos.

Para cada par R^u e P^u, seja $Y_{R^u,P^u} : F^l \to [0,1]$ a função

$$Y_{R^u,P^u}(\overline{R^u}) = \Pr\left[\overline{\mathbf{R}^u} = \overline{R^u} \mid \mathbf{R}^u = R^u, \mathbf{P}^u = P^u, B^u\right]$$

e $Z_{R^u,P^u} : F^l \to [0,1]$ a função

$$Z_{R^u,P^u}(\overline{R^u}) = \Pr\left[\overline{\mathbf{R}^u} = \overline{R^u} \mid \mathbf{R}^u = R^u, \mathbf{P}^u = P^u, \neg B^u\right].$$

Vale a seguinte relação:

$$\begin{aligned}
X_{R^u,P^u}(\overline{R^u}) &= \Pr\left[\overline{\mathbf{R}^u} = \overline{R^u} \mid \mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] \\
&= \Pr\left[B_u, \overline{\mathbf{R}^u} = \overline{R^u} \mid \mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] + \\
&\quad \Pr\left[\neg B_u, \overline{\mathbf{R}^u} = \overline{R^u} \mid \mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] \\
&= \Pr\left[B_u \mid \mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] \Pr\left[\overline{\mathbf{R}^u} = \overline{R^u} \mid \mathbf{R}^u = R^u, \mathbf{P}^u = P^u, B_u\right] + \\
&\quad \Pr\left[\neg B_u \mid \mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] \Pr\left[\overline{\mathbf{R}^u} = \overline{R^u} \mid \mathbf{R}^u = R^u, \mathbf{P}^u = P^u, \neg B_u\right].
\end{aligned}$$

Daí, concluímos que

$$X_{R^u,P^u} = \lambda_u Y_{R^u,P^u} + (1-\lambda_u) Z_{R^u,P^u}$$

para $\lambda_u = \Pr\left[B_u | \mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right]$.

Vamos mostrar agora que para todo par R^u, P^u o número $(1 - nm2^{-k})2^{kl}$ é um limitante inferior para o número de conjuntos de registradores $\overline{R^u}$ para os quais $Y_{R^u,P^u}(\overline{R^u}) = 0$ (ou seja, o número de matrizes $\overline{R^u}$ que não podem estar em $\overline{\mathbf{R}^u}$ quando B^u ocorre). Tal limitante será importante para que possamos limitar posteriormente o valor dos termos $\theta_{F^l}(X_{R^u,P^u})$ que aparecem na equação (6.10).

Fixados R^u, P^u e i, vamos supor que o evento A_{i,u_i} ocorre. Como vimos na prova do teorema 4.3, R_{i,u_i} pode assumir no máximo n elementos de F que não levam o verificador a rejeitar o polinômio nesse caso no teste de consistência (dois polinômios de grau no máximo n distintos coincidem em no máximo n pontos do domínio). Logo, o número de conjuntos $\overline{R^u}$ para os quais

$$\Pr\left[\overline{\mathbf{R}^u} = \overline{R^u} | \mathbf{R}^u = R^u, \mathbf{P}^u = P^u, A_{i,u_i}\right] \neq 0$$

é no máximo $n2^{k(l-1)}$ (são n escolhas possíveis para R_{i,u_i} e $2^{k(l-1)}$ escolhas possíveis para as demais posições). Como temos m valores de i, uma cota superior para o número de conjuntos $\overline{R^u}$ para os quais $Y_{R^u,P^u}(\overline{R^u}) \neq 0$ é $nm2^{k(l-1)}$. Essa é uma cota superior para o número de conjuntos que podem compor $\overline{\mathbf{R}^u}$ quando ocorre o evento B_u.

Agora, vamos limitar $\Pr[B_u]$ assumindo que u foi sorteado aleatoriamente de maneira uniforme. Em cada linha, podemos assumir que o provador "trapaceia" pelo menos uma vez (ou seja, uma trapaça ocorre em pelo menos uma posição em cada linha).

Como cada linha tem N elementos, a probabilidade de o verificador não sortear uma posição em que uma trapaça ocorre em cada uma delas é no máximo $\frac{N-1}{N}$. Logo, a probabilidade de o evento B_u ocorrer é dada por

$$\Pr[B_u] \geq 1 - \left(1 - \frac{1}{N}\right)^m > 1 - e^{-m/N}, \tag{6.11}$$

pois $1 - \frac{1}{N} < e^{-N}$.

Apresentamos agora um lema que será útil na parte final da prova.

Lema 6.2. *Sejam $f, g : S \to [0,1]$ funções tais que $\sum_{s\in S} f(s) \leq 1$ e $\sum_{s\in S} g(s) \leq 1$. Seja $\lambda \in [0,1]$ e seja $r = |\{s \in S | f(s) = 0\}|$. Então $\theta_S(\lambda f + (1-\lambda)g) \leq \sqrt{(1-\lambda)r} + \sqrt{|S| - r}$.*

Demonstração. Inicialmente, observamos que, para toda função $h : S \to [0,1]$ tal que $\sum_{s\in S} h(s) \leq 1$, temos que $\theta_S(h) = \sum_{s\in S} \sqrt{h(s)} \leq \sqrt{|S|}$. De fato, a somatória $\sum_{s\in S} h(s)$ é máxima quando $h(s) = \frac{1}{|S|}$ para todo s em S, e nesse caso claramente a somatória é igual a $\sqrt{|S|}$. Definindo $T = \{s \in S \mid f(s) = 0\}$, temos que

$$\begin{aligned}
\theta_S(\lambda f + (1-\lambda)g) &= \sum_{s\in S} \sqrt{(\lambda f + (1-\lambda)g)(s)} \\
&= \sum_{s\in T} \sqrt{(1-\lambda)g(s)} + \sum_{s\in S\setminus T} \sqrt{(\lambda f + (1-\lambda)g(s)} \\
&= \sqrt{(1-\lambda)}\theta_T(g) + \theta_{S\setminus T}(\lambda f + (1-\lambda)g) \\
&\leq \sqrt{(1-\lambda)|T|} + \sqrt{|S \setminus T|} \\
&= \sqrt{(1-\lambda)r} + \sqrt{|S| - r},
\end{aligned}$$

como queríamos, já que $r = |T|$. □

Utilizando o lema acima, temos que

$$\begin{aligned}
2^{-lk}(\theta_{F^l}(X_{R^u,P^u}))^2 &= 2^{-lk}(\theta_{F^l}(\lambda_u Y_{R^u,P^u} + (1-\lambda_u)Z_{R^u,P^u}))^2 \\
&\leq 2^{-lk}\left(\sqrt{(1-\lambda_u)(2^{lk} - nm2^{k(l-1)})} + \sqrt{nm2^{k(l-1)}}\right)^2 \\
&= 2^{-lk}\left((1-\lambda_u)(2^{lk} - nm2^{k(l-1)}) + nm2^{k(l-1)}\right) + \\
&\quad 2^{-lk}\left(2\sqrt{(1-\lambda_u)(2^{lk} - nm2^{k(l-1)})nm2^{k(l-1)}}\right) \\
&\leq 1 - \lambda_u + 2^{-k}nm\lambda_u + 2\sqrt{(1-\lambda_u)(2^{-lk} - nm2^{-k(l+1)})nm2^{k(l-1)}} \\
&\leq 1 - \lambda_u + 2^{-k}nm\lambda_u + 2\sqrt{(1-\lambda_u)(nm2^{-k} - n^2m^2 2^{-2k})} \\
&\leq 1 - \lambda_u(1 - nm2^{-k}) + 2\sqrt{(1-\lambda_u)nm2^{-k}(1 - nm2^{-k})} \\
&\leq 1 - \Pr\left[B_u | \mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right](1 - nm2^{-k}) + 2\sqrt{nm2^{-k}}.
\end{aligned}$$

Juntando-se (6.9), (6.10) e o resultado acima, temos que a probabilidade de

o verificador ser ludibriado é limitada por

$$
\begin{aligned}
& 2^{-lk} \sum_{R^u, P^u} \Pr\left[\mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] \left(\theta_{F^l}(X_{R^u, P^u})\right)^2 \\
\leq & \sum_{R^u, P^u} \Pr\left[\mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] \\
& \left(1 - \Pr\left[B_u | \mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] \left(1 - nm2^{-k}\right) + 2\sqrt{nm2^{-k}}\right) \\
= & \left(1 + 2\sqrt{nm2^{-k}}\right) \sum_{R^u, P^u} \Pr\left[\mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] - \\
& \left(1 - nm2^{-k}\right) \sum_{R^u, P^u} \Pr\left[\mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] \Pr\left[B_u | \mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] \\
= & \left(1 + 2\sqrt{nm2^{-k}}\right) - \left(1 - nm2^{-k}\right) \sum_{R^u, P^u} \Pr\left[B_u, \mathbf{R}^u = R^u, \mathbf{P}^u = P^u\right] \\
= & 1 + 2\sqrt{nm2^{-k}} - \left(1 - nm2^{-k}\right) \Pr\left[B_u\right] \\
\leq & 1 - (1 - e^{-m/N})(1 - nm2^{-k}) + 2\sqrt{nm2^{-k}},
\end{aligned}
$$

onde a última desigualdade vale por (6.11). Escolhendo m e k como sendo polinômios suficientemente grandes em função do tamanho da entrada n, (por exemplo, $m = (n+1)N$ e $k = 2n + dm + 6$), a probabilidade de o provador ser ludibriado é menor que 2^{-n}. □

Capítulo 7

Simulações de Sistemas Interativos Quânticos

Nesse capítulo, apresentamos alguns resultados de Kitaev e Watrous [KW00] envolvendo simulações de **SIQP**s por outros **SIQP**s, de modo a reduzir o número de trocas de mensagens necessárias para a decisão de linguagens.

Inicialmente mostramos como um **SIQP** em que tanto a aceitação como a rejeição são feitas com alguma probabilidade de erro pode ser simulada por outro **SIQP** em que as rejeições sempre são feitas corretamente. Por fim, mostramos que todo **SIQP** que utiliza uma quantidade polinomial de mensagens pode ser simulado por outro **SIQP** que realiza apenas três trocas de mensagens. Este último resultado compõe uma forte evidência da superioridade do modelo quântico de computação em relação ao clássico, para o qual não se conhece ainda resultados análogos.

7.1 Eliminação de rejeição incorreta

Nessa seção, mostramos que se temos um **SIQP** que tanto aceita como rejeita erroneamente, mas com alguma delimitação nas probabilidades de resposta errada, então podemos montar um novo sistema que nunca devolve uma rejeição incorreta utilizando o protocolo original e duas mensagens adicionais. Para mostrar tal resultado, vamos primeiro apresentar uma definição estendida dos sistemas interativos quânticos de prova.

Para funções $m : \mathbb{Z}^+ \to \mathbb{N}$ e $a, b : \mathbb{Z}^+ \to [0, 1]$, denotamos por $\mathbf{QIP}(m, a, b)$ a classe de linguagens L para as quais existe um verificador V de m mensagens tal que

1. Existe um provador P de m mensagens tal que, para qualquer x em L,

(P, V) aceita x com probabilidade limitada inferiormente por $a(|x|)$.

2. Para qualquer provador P de m mensagens e para qualquer $x \notin L$, (P, V) aceita x com probabilidade limitada superiormente por $b(|x|)$.

Apresentamos agora o resultado principal dessa seção, seguindo a exposição de Kitaev e Watrous [KW00].

Teorema 7.1. *Seja m um polinômio em uma variável e seja $a : \mathbb{Z}^+ \to [0, 1]$ uma função para a qual existe uma família uniformemente polinomial de circuitos $\{Q_{1^n}\}$ tal que Q_{1^n} calcula a transformação unitária $T_{a(n)}$, dada por*

$$T_{a(n)}(|0\rangle) = \sqrt{a(n)}|0\rangle - \sqrt{1 - a(n)}|1\rangle,$$
$$T_{a(n)}(|1\rangle) = \sqrt{1 - a(n)}|0\rangle - \sqrt{a(n)}|1\rangle.$$

Então, para $b : \mathbb{Z}^+ \to [0, 1]$ tal que $b(n) < a(n)$ para todo n, $\mathbf{QIP}(m, a, b) \subseteq \mathbf{QIP}(m + 2, 1, 1 - (a - b)^2)$.

Demonstração. Seja L uma linguagem de $\mathbf{QIP}(m, a, b)$. Pela definição, existe um verificador de m mensagens e um provador de m mensagens tais que o provador faz com que o verificador aceite uma entrada x, onde $x \in L$ e $|x| = n$, com probabilidade limitada inferiormente por $a(n)$. Vamos assumir que o verificador e o provador são tais que a probabilidade de o verificador aceitar x se $x \in L$ é exatamente igual a $a(n)$. Caso essa probabilidade seja $a(n)' \geq a(n)$, basta que o verificador, ao atingir um estado de aceitação, faça um sorteio aleatório em que $|0\rangle$ é obtido com probabilidade $a(n)/a(n)'$ e $|1\rangle$ com probabilidade $1 - a(n)/a(n)'$. Se o valor obtido após esse sorteio for $|0\rangle$, ele aceita a entrada, senão ele rejeita.

Para mostrar que L é uma linguagem de $\mathbf{QIP}(m + 2, 1, 1 - (a - b)^2)$, utilizamos o protocolo cujo algoritmo do verificador descrevemos a seguir.

Vamos denotar por $\mathbf{R}$ o conjunto de registradores utilizados pelo verificador do protocolo original. Sejam $\mathbf{B}$ e $\mathbf{B}$' dois registradores que serão utilizados pelo verificador do novo protocolo. Os registradores $\mathbf{B}$ e $\mathbf{B}$' são inicializados com $|0\rangle$ e naturalmente não são alterados pelo protocolo original.

Primeiramente, o protocolo original é executado, mas o verificador não devolve a resposta. Suponha que $|\psi\rangle = \alpha_{acc}|\psi_{acc}\rangle + \alpha_{rej}|\psi_{rej}\rangle$ descreve o estado de $\mathbf{R}$ e dos registradores privados do provador original após a execução do protocolo original, onde $|\psi_{acc}\rangle$ e $|\psi_{rej}\rangle$ representam, respectivamente, as projeções normalizadas de $|\psi\rangle$ nos estados de aceitação e rejeição do protocolo original.

Nesse momento, o verificador incrementa os registradores $\mathbf{B}$ e $\mathbf{B}$' nos estados básicos armazenados em $\mathbf{R}$ que levam o verificador a rejeitar a entrada no

protocolo original (ou seja, nos estados que estão na superposição $|\psi_{rej}\rangle$). Após o incremento de $\mathbf{B}$ e $\mathbf{B}$', o estado do sistema será

$$|\psi\rangle = \alpha_{acc}|00\rangle|\psi_{acc}\rangle + \alpha_{rej}|11\rangle|\psi_{rej}\rangle.$$

Feito isso, o verificador envia os registradores $\mathbf{B}$' e $\mathbf{R}$ ao provador novo. Ao recebê-los, o provador realiza alguma transformação unitária U nesses qubits. O estado do sistema passa a ser

$$|\psi'\rangle = \alpha_{acc}|0\rangle U(|0\rangle|\psi_{acc}\rangle) + \alpha_{rej}|1\rangle U(|1\rangle|\psi_{rej}\rangle).$$

Após aplicar essa operação, o provador envia $\mathbf{B}$' ao verificador, que realiza a operação $\mathbf{B'} = \mathbf{B'} - \mathbf{B}$. O estado do sistema passa a ser

$$|\psi''\rangle = \alpha_{acc}|0\rangle|\phi_{acc}\rangle + \alpha_{rej}|1\rangle|\phi_{rej}\rangle,$$

onde $|\phi_{acc}\rangle$ é o estado $U(|0\rangle|\psi_{acc}\rangle)$ e $|\phi_{rej}\rangle$ é o estado $U(|1\rangle|\psi_{rej}\rangle)$, mas com $\mathbf{B'}$ já alterado pela operação $\mathbf{B'} = \mathbf{B'} - \mathbf{B}$ (ou seja, com $\mathbf{B'}$ complementado).

Por fim, o verificador aplica $T_{a(n)}$ em $\mathbf{B}$. Denotando por $||\phi\rangle|$ o número de qubits do registrador que contém $|\phi\rangle$, o estado obtido será

$$\begin{aligned} T_{a(n)} \otimes I_{||\phi\rangle|}(|\psi''\rangle) \;\; = \;\; & \alpha_{acc}\left(\sqrt{a(n)}|0\rangle|\phi_{acc}\rangle + \sqrt{1-a(n)}|1\rangle|\phi_{acc}\rangle\right) + \\ & \alpha_{rej}\left(\sqrt{1-a(n)}|0\rangle|\phi_{rej}\rangle + \sqrt{a(n)}|1\rangle|\phi_{rej}\rangle\right). \end{aligned}$$

Nesse estado, o verificador mede $\mathbf{B}$. Se o conteúdo obtido for $|0\rangle$, ele aceita a entrada, e se for $|1\rangle$, rejeita. Logo, a probabilidade de o verificador aceitar a entrada x é dada por

$$\left\| \alpha_{acc}\sqrt{a(n)}|\phi_{acc}\rangle + \alpha_{rej}\sqrt{1-a(n)}|\phi_{rej}\rangle \right\|^2. \tag{7.1}$$

Suponha que $x \in L$. Como $L \in \mathbf{QIP}(m, a, b)$, temos que $\alpha_{acc} = \sqrt{a(n)}$ e $\alpha_{rej} = \sqrt{1-a(n)}$. Nesse caso, uma estratégia adequada para o provador seria escolher uma transformação U tal que $U(|0\rangle|\psi_{acc}\rangle) = |0\rangle|\gamma\rangle$ e $U(|1\rangle|\psi_{rej}\rangle) = |1\rangle|\gamma\rangle$, onde $|\gamma\rangle$ é um estado arbitrário. Destacamos que U não altera o conteúdo de $\mathbf{B}$'. Dessa maneira, $|\phi_{acc}\rangle = |\phi_{rej}\rangle$, pois $\mathbf{B'}$ passa a ser zero nos dois estado após a operação $\mathbf{B'} = \mathbf{B'} - \mathbf{B}$, e a probabilidade (7.1) assume o valor 1, como queríamos.

Suponha agora que $x \notin L$. Lembrando que a média geométrica é menor ou igual que a média aritmética e que $\alpha_{rej} = \sqrt{1-\alpha_{acc}^2}$, temos que a probabilidade

de o verificador aceitar a entrada é dada por

$$\begin{aligned}
&\left(\alpha_{acc}\sqrt{a(n)} + \alpha_{rej}\sqrt{1-a(n)}\right)^2 \\
&= \alpha_{acc}^2 a(n) + 2\alpha_{acc}\alpha_{rej}\sqrt{a(n)-a(n)^2} + \alpha_{rej}^2(1-a(n)) \\
&= \alpha_{acc}^2 a(n) + 2\alpha_{acc}\sqrt{1-\alpha_{acc}^2}\sqrt{a(n)-a(n)^2} + (1-\alpha_{acc}^2)(1-a(n)) \\
&= \alpha_{acc}^2 a(n) + 2\alpha_{acc}\sqrt{1-\alpha_{acc}^2}\sqrt{a(n)-a(n)^2} + 1 - a(n) - \alpha_{acc}^2 + \alpha_{acc}^2 a(n) \\
&= 2\alpha_{acc}^2 a(n) + 2\sqrt{\alpha_{acc}^2 - \alpha_{acc}^4}\sqrt{a(n)-a(n)^2} + 1 - a(n) - \alpha_{acc}^2 \\
&\leq 2\alpha_{acc}^2 a(n) + \frac{2\left(\alpha_{acc}^2 - \alpha_{acc}^4 + a(n) - a(n)^2\right)}{2} + 1 - a(n) - \alpha_{acc}^2 \\
&= 1 - a(n)^2 + 2\alpha_{acc}^2 a(n) - \alpha_{acc}^4 \\
&= 1 - \left(a(n) - \alpha_{acc}^2\right)^2.
\end{aligned}$$

Como $\alpha_{acc} \leq \sqrt{b(n)}$, temos que a probabilidade acima é limitada por $1-(a(n)-b(n))^2$, como queríamos. Logo, $\mathbf{QIP}(m,a,b) \subseteq \mathbf{QIP}(m+2,1,1-(a-b)^2)$. □

7.2 Paralelização de SIQPs

O objetivo dessa seção é mostrar que um **SIQP** que utiliza uma quantidade polinomial de trocas de mensagens pode ser simulado por outro que utiliza apenas três trocas de mensagens. Antes, apresentamos algumas definições que serão úteis nessa demonstração.

Uma *mistura* é uma coleção $\{(p_k, |\phi_k\rangle)\}$ definida num espaço $\mathcal{H}_t$ onde os valores p_k são números reais não-negativos tais que $\sum_k p_k = 1$ e cada $|\phi_k\rangle$ é uma superposição de $\mathcal{H}_t$. Associamos a uma mistura um *operador de densidade* (ou *matriz de densidade*) $\rho = \sum_k p_k |\phi_k\rangle\langle\phi_k|$. O conjunto dos operadores de densidade num espaço de Hilbert $\mathcal{H}$ é denotado por $\mathbf{D}(\mathcal{H})$. Esses operadores de densidade são sempre semi-definidas e têm traço unitário.

Os operadores de densidade são muito convenientes quando queremos estudar sistemas quânticos cujos estados não são completamente conhecidos. Porém, como formulação matemática, o operador de densidade é equivalente ao vetor de superposições ([NC00]). Mais ainda: para cada operador de densidade ρ, existe uma superposição $|\psi\rangle$ tal que $\rho = |\psi\rangle\langle\psi|$.

Uma operação importante envolvendo operadores de densidade é a de *traço-remoção*, definida da seguinte maneira: dado um operador de densidade $\rho \in \mathbf{D}(\mathcal{H} \otimes \mathcal{K})$, uma base ortonormal qualquer $\{|e_1\rangle, \ldots, |e_n\rangle\}$ de $\mathcal{K}$ e denotando por $I_{\mathcal{H}}$ a matriz identidade de dimensão igual a do espaço $\mathcal{H}$, a operação de

traço-remoção é dada por

$$\mathrm{tr}_{\mathcal{K}}\,\rho = \sum_{j=1}^{n} (I_{\mathcal{H}} \otimes \langle e_j|)\rho(I_{\mathcal{H}} \otimes |e_j\rangle).$$

Esta operação é importante porque eventualmente estamos interessados somente em um subconjunto dos qubits de um sistema quântico. Se ρ é um operador de densidade definido num espaço $\mathcal{H} \otimes \mathcal{K}$, então $\mathrm{tr}_{\mathcal{K}}\,\rho$ é o operador de densidade em $\mathcal{H}$ associado a ρ quando ignoramos os qubits do espaço $\mathcal{K}$.

Apresentamos agora um exemplo que mostra a relação entre superposições e misturas, bem como um caso de uso da operação de traço-remoção. Considere o estado $|\phi\rangle = \frac{1}{\sqrt{2}}|000\rangle + \frac{1}{2}|001\rangle + \frac{1}{2}|010\rangle$. Tal estado pode ser descrito pela mistura $\{(\frac{1}{2}, |000\rangle), (\frac{1}{4}, |001\rangle), (\frac{1}{4}, |010\rangle)\}$, onde cada um dos estados puros também é um estado básico. O operador de densidade associado a essa mistura é dado abaixo:

$$\rho = \sum_{k\in\{0,1\}^3} \rho_k |k\rangle\langle k| = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Vamos realizar agora a operação de traço-remoção do terceiro qubit de $|\phi\rangle$.

Como esse qubit está em $\mathcal{H}_2$, seguindo a definição, temos o seguinte:

$$
\begin{aligned}
\operatorname{tr}_{\mathcal{H}_2}\rho &= \sum_{k\in\{0,1\}} (I_{\mathcal{H}_4}\otimes\langle k|)\,\rho\,(I_{\mathcal{H}_4}\otimes|k\rangle) \\
&= I_4\otimes\begin{pmatrix}1&0\end{pmatrix}\ \rho\ I_4\otimes\begin{pmatrix}1\\0\end{pmatrix} + I_4\otimes\begin{pmatrix}0&1\end{pmatrix}\ \rho\ I_4\otimes\begin{pmatrix}0\\1\end{pmatrix} \\
&= \begin{pmatrix}1&0&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&0&1&0\end{pmatrix}\ \rho\ \begin{pmatrix}1&0&0&0\\0&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&0\\0&0&0&1\\0&0&0&0\end{pmatrix} + \\
&\quad \begin{pmatrix}0&1&0&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&1\end{pmatrix}\ \rho\ \begin{pmatrix}0&0&0&0\\1&0&0&0\\0&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&0\\0&0&0&1\end{pmatrix} \\
&= \begin{pmatrix}\frac{1}{2}&0&0&0\\0&\frac{1}{4}&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix} + \begin{pmatrix}\frac{1}{4}&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix} \\
&= \begin{pmatrix}\frac{3}{4}&0&0&0\\0&\frac{1}{4}&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}.
\end{aligned}
$$

Por fim, dizemos que $F(\rho,\xi) = \left\|\sqrt{\rho}\sqrt{\xi}\right\|_{\mathrm{tr}}^2$ é a *fidelidade* entre dois operadores de densidade ρ e ξ definidos num mesmo espaço de Hilbert, onde $\|X\|_{\mathrm{tr}} = \operatorname{tr}\sqrt{X^\dagger X}$. Uma matriz B é raiz de uma matriz T se $T = BB$. A raiz de uma matrizes de densidade sempre está definida, pois ela é hermitiana.

Partimos agora para a demonstração do resultado principal dessa seção.

Teorema 7.2. *Seja m um polinômio e seja $\epsilon : \mathbb{Z}^+ \to [0,1]$ uma função qualquer. Então* $\mathbf{QIP}(m,1,1-\epsilon) \subseteq \mathbf{QIP}(3,1,1-\frac{\epsilon^2}{4m^2})$.

Demonstração. Seja $x \in \Sigma^*$ e L uma linguagem em $\mathbf{QIP}(m, 1, 1-\epsilon)$. Vamos apresentar um protocolo com o qual L é decidida utilizando três trocas de mensagem. Para isso, vamos utilizar como base o protocolo de m trocas de mensagens que por hipótese já é conhecido. Tal protocolo será referenciado daqui para frente como o protocolo *original*.

Assumimos sem perda de generalidade que m é ímpar. Logo, o provador é o primeiro a enviar uma mensagem no protocolo original. Se no protocolo original o valor de m era par, acrescentamos uma primeira mensagem enviada pelo provador, que é composta por um único qubit no estado $|0\rangle$. Ajustamos também o verificador para que rejeite a prova se o qubit enviado pelo provador na primeira mensagem, ao ser medido, não resultar em 0. Assumimos assim que $m \geq 5$. Dessa forma, temos que $k = (m+1)/2$ é o número de circuitos utilizados pelo provador e pelo verificador.

Como $L \in \mathbf{QIP}(m, 1, 1-\epsilon)$, existe uma seqüência de circuitos $V_1, \ldots, V_k$ do verificador tal que

1. se $x \in L$, existe uma seqüência de circuitos $P_1, \ldots, P_k$ do provador tal que (P, V) sempre aceita x.

2. se $x \notin L$, para qualquer seqüência de circuitos $P_1, \ldots, P_k$ do provador, (P, V) aceita x com probabilidade limitada superiormente por $1 - \epsilon$.

O protocolo original utiliza os seguintes registradores: o registrador $\mathbf{V}$ de qubits privados do verificador; o registrador $\mathbf{P}$ de qubits privados do provador; e o registrador $\mathbf{M}$ de qubits de comunicação do sistema. Denotamos por $\mathcal{V}$ o espaço de Hilbert no qual estão contidos os vetores que descrevem o registrador $\mathbf{V}$, por $\mathcal{P}$ o espaço correspondente ao registrador $\mathbf{P}$ e por $\mathcal{M}$ o espaço correspondente ao registrador $\mathbf{M}$. Por fim, denotamos por 2^v, 2^m e 2^p as dimensões de $\mathcal{V}$, $\mathcal{M}$ e $\mathcal{P}$, respectivamente.

Tanto os circuitos $V_1', \ldots, V_k'$ como os circuitos $P_1', \ldots, P_k'$ são operadores no espaço $\mathcal{V} \otimes \mathcal{M} \otimes \mathcal{P}$. Porém, cada circuito V_i' atua sobre os registradores $\mathbf{V}$ e $\mathbf{M}$ e mantém o conteúdo dos registradores de $\mathbf{P}$ inalterados, enquanto cada circuito P_i atua sobre os registradores $\mathbf{P}$ e $\mathbf{M}$ e mantém o conteúdo dos registradores de $\mathbf{V}$ inalterados. Assim, a rigor temos que cada V_i' pode ser escrito como um operador $V_i \otimes I_{2^p}$ e que cada P_i' pode ser escrito como um operador $I_{2^v} \otimes P_i$.

Denotamos por Π'_{init} a projeção de um vetor de $\mathcal{V} \otimes \mathcal{M} \otimes \mathcal{P}$ em outro vetor tal que os qubits de $\mathcal{V}$ são todos iguais a $|0\rangle$ (ou seja, corresponde à projeção do vetor no subespaço gerado pelos elementos da base de $\mathcal{V} \otimes \mathcal{M} \otimes \mathcal{P}$ onde as coordenadas referentes ao espaço $\mathcal{V}$ são iguais a zero). O vetor resultante dessa projeção será denotado por $|\psi_{init}\rangle$.

Denotamos por Π'_{acc} a projeção de um vetor de $\mathcal{V} \otimes \mathcal{M} \otimes \mathcal{P}$ em outro vetor que é uma combinação linear de um subconjunto de elementos da base

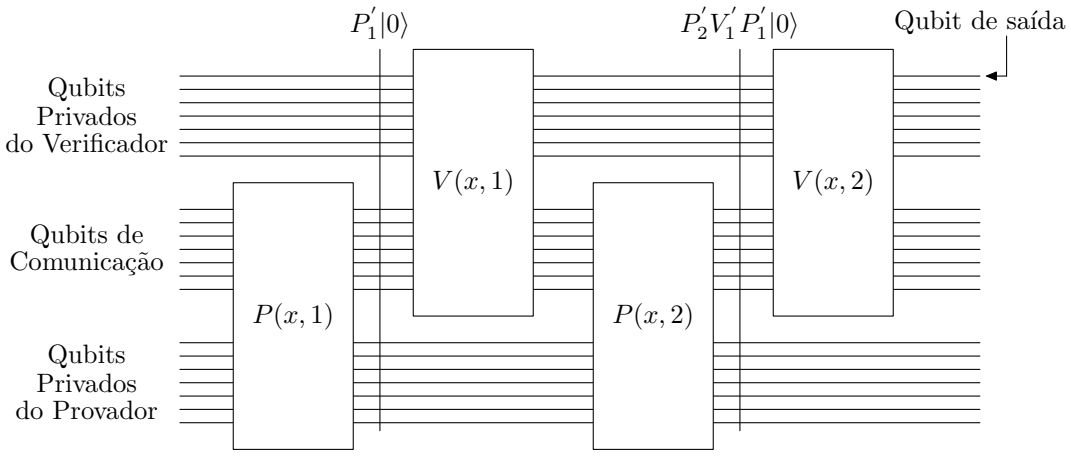


Figura 7.1: Esquema de aplicação dos circuitos do provador e do verificador.

de $\mathcal{V} \otimes \mathcal{M} \otimes \mathcal{P}$ chamados de estados de aceitação. Ou seja, é uma projeção nos estados em que o qubit de saída é 1.

Na prática, tanto Π'_{init} como Π'_{acc} são aplicadas apenas nos registradores que compõem o espaço $\mathcal{V} \otimes \mathcal{M}$, mantendo os registradores correspondentes a $\mathcal{P}$ inalterados. Dessa forma, podemos reescrever as duas projeções na forma $\Pi'_{acc} = \Pi_{acc} \otimes I_{2^p}$ e $\Pi'_{init} = \Pi_{init} \otimes I_{2^p}$.

O protocolo em questão utiliza três conjuntos de registradores $\mathbf{V}_1, \ldots, \mathbf{V}_k$, $\mathbf{M}_1, \ldots, \mathbf{M}_k$ e $\mathbf{P}_1, \ldots, \mathbf{P}_k$. A idéia é que os registradores $\mathbf{V}_i, \mathbf{M}_i$ e $\mathbf{P}_i$ servirão para armazenar o vetor $P_i' V_{i-1}' P_{i-1}' \ldots V_1' P_1' |0\rangle$. A figura 7.2 ilustra a idéia.

Com isso, já introduzimos a notação necessária para apresentar o protocolo do verificador.

O protocolo e a aceitação de instâncias verdadeiras

Vamos agora descrever em detalhes o funcionamento do protocolo, mostrando que se x está em L, então existe um provador que faz o verificador aceitar a prova sempre. Nas figuras 7.3 e 7.2, apresentamos os algoritmos do verificador e do provador, respectivamente, para mostrar o funcionamento do protocolo.

Nesse protocolo, as operações são realizadas num espaço de dimensão $2^{k(v+m+p)+1+\lceil \lg(k-1) \rceil}$. Isso porque temos k registradores de $\mathcal{V}$, de $\mathcal{M}$ e de $\mathcal{P}$, um qubit B e um registrador de $\lceil \lg(k-1) \rceil$ qubits (que vai ser utilizado para armazenar um número r). Aqui, B e r são registradores privados do verificador e os k registradores de $\mathcal{P}$ são privados do provador. Os demais são registradores de comunicação.

Quando nos referirmos a uma operação num espaço de dimensão inferior a

Algoritmo do Provador

01 $\{\mathbf{V}_1, \mathbf{M}_1, \mathbf{P}_1\} \leftarrow P_1|\psi_{init}\rangle$

02 Para i de 1 até $k-1$ faça

03 $\quad\{\mathbf{V}_{i+1}, \mathbf{M}_{i+1}, \mathbf{P}_{i+1}\} \leftarrow P_{i+1}V_iP_i \ldots V_1P_1|\psi_{init}\rangle$

04 **Envia** $\mathbf{V}_1, \ldots, \mathbf{V}_k$ e $\mathbf{M}_1, \ldots, \mathbf{M}_k$ para o verificador.

05 **Recebe** $\mathbf{M}_r, \mathbf{M}_{r+1}, B'$ e r do verificador

06 Aplica P_{r+1} em $(\mathbf{M}_r, \mathbf{P}_r)$

07 Aplica troca controlada em $((\mathbf{M}_r, \mathbf{P}_r), (\mathbf{M}_{r+1}, \mathbf{P}_{r+1}), B')$

08 **Envia** B' para o verificador

Figura 7.2: Algoritmo do provador.

$2^{k(v+m+p)+1+\lceil \lg(k-1)\rceil}$, o que estamos fazendo na verdade é a omissão da operação de produto tensorial com matrizes identidade de dimensão adequada.

O provador começa inicializando os registradores $\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_k$ e $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_k$. Para isso, ele inicializa cada um destes registradores com zero e simula o protocolo original utilizando as seqüências de transformações (i.e., as seqüências de circuitos) $P_1', \ldots, P_k'$ e $V_1', \ldots, V_k'$ que levam o verificador original a aceitar a entrada sempre. Mais exatamente, ele inicializa $\mathbf{V}_i, \mathbf{M}_i$ e $\mathbf{P}_i$ com $P_i'V_{i-1}'P_{i-1}' \ldots V_1'P_1'|0\rangle$.

O verificador por sua vez começa a fazer suas checagens após receber a mensagem do provador, que contém os registradores $\mathbf{V}_1, \ldots, \mathbf{V}_k$ e $\mathbf{M}_1, \ldots, \mathbf{M}_k$. Inicialmente, ele faz a medição de $(\mathbf{V}_1, \mathbf{M}_1)$ para checar se $\mathbf{V}_1 = 0$. Logo em seguida, o verificador aplica V_k em $(\mathbf{V}_k, \mathbf{M}_k)$ e mede o qubit de saída para checar se ele é 1.

Se em alguma dessas medições o verificador não obtém o resultado esperado (ou seja, se na primeira medição ele obteve um valor diferente de 0 ou se na última ele obteve um valor diferente de 1), a prova é rejeitada. Esse não será o caso se o provador inicializou registradores como descrito anteriormente.

Ao final dessa primeira fase de testes, o verificador aplica $V_k^\dagger$ em $(\mathbf{V}_k, \mathbf{M}_k)$. Essa operação é a inversa de V_k, e quando é aplicada no par de registradores $(\mathbf{V}_k, \mathbf{M}_k)$ (que tinham sido alterados após a aplicação de V_k), faz com que os dois registradores voltem ao que eram anteriormente (no momento em que o provador os enviou para o verificador, a menos das medições).

Assim, se a entrada não é rejeitada, as duas primeiras checagens do verificador correspondem à aplicação das projeções Π_{init} em $(\mathbf{V}_1, \mathbf{M}_1)$ e à aplicação de V_k, seguida de Π_{acc} e seguida de $V_k^\dagger$ em $(\mathbf{V}_k, \mathbf{M}_k)$. O efeito é deixar o sistema no estado $|\psi_{init}\rangle$.

Algoritmo do Verificador

01 **Recebe** $\mathbf{V}_1, \ldots, \mathbf{V}_k$ e $\mathbf{M}_1, \ldots, \mathbf{M}_k$ do provador.
02 Se algum qubit de $\mathbf{V}_1$ é diferente de zero
03 Rejeita a prova
04 Aplica V_k em $(\mathbf{V}_k, \mathbf{M}_k)$
05 Se $(\mathbf{V}_k, \mathbf{M}_k)$ não contém um estado em que o qubit de saída é igual a 1
06 Rejeita a prova
07 Aplica $V_k^\dagger$ em $(\mathbf{V}_k, \mathbf{M}_k)$
08 Sorteia r em $\{1, \ldots, k-1\}$
09 Prepara $(B, B') = \frac{1}{\sqrt{2}}(|0\rangle|0\rangle + |1\rangle|1\rangle)$
10 Aplica V_r em $(\mathbf{V}_r, \mathbf{M}_r)$
11 Aplica troca controlada em $(\mathbf{V}_r, \mathbf{V}_{r+1}, B)$
12 **Envia** $\mathbf{M}_r, \mathbf{M}_{r+1}, B'$ e r para o provador
13 **Recebe** B' do provador
14 Aplica negação controlada em (B, B')
15 Aplica Hadamard em B
16 Se B for diferente de zero
17 Rejeita a prova
18 Aceita a prova

Figura 7.3: Algoritmo do verificador.

Em seguida, o verificador sorteia aleatoriamente um número r no conjunto $\{1, 2, \ldots, k-1\}$ e inicializa o qubit B no estado $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ (que até esse ponto estava no estado $|0\rangle$), utilizando a transformação de Hadamard em B. Feito isso, o verificador aplica V_r em $(\mathbf{V}_r, \mathbf{M}_r)$.

Após realizar essa transformação, o verificador faz uma *troca controlada* entre $\mathbf{V}_r$ e $\mathbf{V}_{r+1}$ utilizando B como controlador ($\mathbf{V}_r$ e $\mathbf{V}_{r+1}$ "trocam" de posição nos estados em que B é $|1\rangle$). Seja $|t\rangle$ o estado de $\mathbf{V}_r$ após a aplicação de V_r, $|y\rangle$ o estado de $\mathbf{V}_{r+1}$ e $|z\rangle$ o estado dos outros registradores do sistema (menos o qubit B) após a execução da linha 11 do algoritmo do verificador. Nesse ponto, o estado do sistema é

$$|\phi\rangle \;=\; \frac{1}{\sqrt{2}}\left(|0\rangle|t\rangle|y\rangle|z\rangle + |1\rangle|y\rangle|t\rangle|z\rangle\right).$$

Se o provador seguiu o esquema acima, a troca realizada na linha 11 não modifica o conteúdo dos registradores $\mathbf{V}_r$ e $\mathbf{V}_{r+1}$ (ou seja, $|t\rangle = |y\rangle$).

Nesse ponto, o verificador envia sua mensagem ao provador. Essa mensagem vai conter $\mathbf{M}_r$ (já modificado por V_r), $\mathbf{M}_{r+1}$, B' e r.

Após receber tais registradores, o provador aplica a transformação P_{r+1} em $(\mathbf{M}_r, \mathbf{P}_r)$. Utilizando o registrador $\mathbf{P}_r$ correto (i.e., se $\mathbf{P}_r$ for o r-ésimo conteúdo dos qubits do provador no protocolo original), $\mathbf{M}_r$ passa a ficar igual a $\mathbf{M}_{r+1}$ e $\mathbf{P}_r$ fica igual a $\mathbf{P}_{r+1}$.

Em seguida, o provador faz uma troca controlada entre $(\mathbf{M}_r, \mathbf{P}_r)$ e $(\mathbf{M}_{r+1}, \mathbf{P}_{r+1})$ utilizando B' como qubit de controle. Tal operação não altera o conteúdo dos registradores nesse caso. Feito isso, o provador envia uma mensagem para o provador contendo o qubit B'.

Após receber B' do provador, o verificador realiza uma negação controlada em (B, B') (o conteúdo de B' é "invertido" nos estados em que B é $|1\rangle$). Como o provador não altera o conteúdo de B', após essa operação B fica no estado $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, e B' fica no estado puro $|0\rangle$.

Por fim, o verificador aplica a transformação de Hadamard em B. Após a aplicação da matriz de Hadamard no qubit B, o estado do sistema passa a ser o vetor $|\phi'\rangle$, dado por

$$\frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}|0\rangle|t\rangle|y\rangle|z\rangle + \frac{1}{\sqrt{2}}|1\rangle|t\rangle|y\rangle|z\rangle\right) + \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}|0\rangle|y\rangle|t\rangle|z\rangle - \frac{1}{\sqrt{2}}|1\rangle|y\rangle|t\rangle|z\rangle\right).$$

Logo, como $|y\rangle = |y\rangle$, o que temos na verdade é $|\phi'\rangle = |0\rangle|y\rangle|y\rangle|z\rangle$ e certamente a prova não é rejeitada nas linhas 13-14 do algoritmo do verificador. Ou seja, no final da interação, o verificador aceita a entrada.

Limitação da probabilidade de erro

Vamos analisar agora o caso em que a entrada x não pertence à linguagem, estabelecendo uma cota superior para a probabilidade de aceitação incorreta pelo verificador no protocolo apresentado. Neste caso, sabemos que, qualquer que seja a seqüência de circuitos $P_1, \ldots, P_k$ utilizada pelo provador, a probabilidade com a qual o verificador aceita x é limitada superiormente por $1 - \epsilon$.

Assim como no artigo de Kitaev e Watrous [KW00], aqui utilizaremos as seguintes proposições. A proposição 7.3 foi provada por Jozsa [Joz94] e a proposição 7.5 foi provada por Hughston et al. [HJW93].

Proposição 7.3. *Sejam $\mathcal{H}$ e $\mathcal{K}$ espaços de Hilbert tais que* $\dim(\mathcal{H}) \leq \dim(\mathcal{K})$ *(onde* $\dim(V)$ *denota a dimensão do espaço V), e sejam $\phi_1, \phi_2 \in \mathbf{D}(\mathcal{H})$ (ou seja, matrizes de densidade definidas em $\mathcal{H}$). Então $F(\phi_1, \phi_2) = \max\{|\langle\xi|\gamma\rangle|^2\}$, onde o máximo é escolhido dentre todos os estados $|\xi\rangle, |\gamma\rangle \in \mathcal{H} \otimes \mathcal{K}$ tais que $\mathrm{tr}_{\mathcal{K}} |\xi\rangle\langle\xi| = \phi_1$ e $\mathrm{tr}_{\mathcal{K}} |\gamma\rangle\langle\gamma| = \phi_2$. Equivalentemente, para $\eta = \min\{\||\xi\rangle - |\gamma\rangle\|\}$, onde o mínimo é escolhido dentre todos os estados $|\xi\rangle, |\gamma\rangle \in \mathcal{H} \otimes \mathcal{K}$ tais que $\mathrm{tr}_{\mathcal{K}} |\xi\rangle\langle\xi| = \phi_1$ e $\mathrm{tr}_{\mathcal{K}} |\gamma\rangle\langle\gamma| = \phi_2$, temos $F(\phi_1, \phi_2) = (1 - \eta^2/2)^2$.* □

Proposição 7.4. *Se $|\phi\rangle, |\psi\rangle \in \mathcal{H} \otimes \mathcal{K}$ são tais que $\mathrm{tr}_{\mathcal{K}} |\phi\rangle\langle\phi| = \mathrm{tr}_{\mathcal{K}} |\psi\rangle\langle\psi|$, então existe um operador unitário U no espaço de Hilbert $\mathcal{K}$ tal que $(I_{\dim(\mathcal{H})} \otimes U)|\phi\rangle = |\psi\rangle$.* □

Proposição 7.5. *Se $|\phi\rangle, |\psi\rangle \in \mathcal{H} \otimes \mathcal{K}$ são tais que $\mathrm{tr}_{\mathcal{K}} |\phi\rangle\langle\phi| = \mathrm{tr}_{\mathcal{K}} |\psi\rangle\langle\psi|$, então existe um operador unitário U no espaço de Hilbert $\mathcal{K}$ tal que $(I \otimes U)|\phi\rangle = |\psi\rangle$.* □

Observe que $\dim(\mathcal{V}) \leq \dim(\mathcal{M} \otimes \mathcal{P})$. Lembrando que k é o número de circuitos nas seqüências do verificador e do provador e que m é o número de trocas de mensagens do protocolo original, como $1 \leq r \leq k - 1$ para $k = (m + 1)/2$, esse sorteio consome $\lceil \lg(k - 1) \rceil$ qubits. Como o verificador do protocolo original tem pelo menos um qubit privado (o qubit de decisão) e como existe pelo menos um qubit de comunicação no protocolo original, temos que o espaço $\mathcal{M}$ tem pelo menos $2k \geq 1 + \lceil \lg(k - 1) \rceil$ qubits, e portanto $\dim(\mathcal{V}) \leq \dim(\mathcal{M} \otimes \mathcal{P})$.

Utilizando essas duas proposições, vamos provar um lema que será útil na análise desse protocolo.

Lema 7.6. *Sejam $\rho_1, \ldots, \rho_k$ elementos de $\mathbf{D}(\mathcal{V} \otimes \mathcal{M})$ tais que $\rho_1 = \Pi_{init} \rho_1 \Pi_{init}$ e $\rho_k = (V_k^\dagger \Pi_{acc} V_k) \rho_k (V_k^\dagger \Pi_{acc} V_k)$. Nesse caso, temos que*

$$\sum_{j=1}^{k-1} \sqrt{F(\mathrm{tr}_{\mathcal{M}} V_j \rho_j V_j^\dagger, \mathrm{tr}_{\mathcal{M}} \rho_{j+1})} \leq k - 1 - \frac{\epsilon^2}{8(k-1)}.$$

Demonstração. Sabemos que $\dim(\mathcal{V}) \leq \dim(\mathcal{M} \otimes \mathcal{P})$ e que tanto $\mathrm{tr}_{\mathcal{M}} V_j \rho_j V_j^\dagger$ como $\mathrm{tr}_{\mathcal{M}} \rho_{j+1}$ são elementos de $\mathbf{D}(\mathcal{V})$. Assim, pela proposição 7.3, para $1 \leq j \leq k-1$, existem vetores $|\xi_j\rangle, |\gamma_j\rangle$ no espaço $\mathcal{V} \otimes \mathcal{M} \otimes \mathcal{P}$ tais que

1. $\mathrm{tr}_{\mathcal{M}\otimes\mathcal{P}} |\xi_j\rangle\langle\xi_j| = \mathrm{tr}_{\mathcal{M}} V_j \rho_j V_j^\dagger$;
2. $\mathrm{tr}_{\mathcal{M}\otimes\mathcal{P}} |\gamma_j\rangle\langle\gamma_j| = \mathrm{tr}_{\mathcal{M}} \rho_{j+1}$;
3. $\||\xi_j\rangle - |\gamma_j\rangle\| = \eta_j$,

onde cada η_j é um número real não-negativo tal que

$$\eta_j = \sqrt{2 - 2\sqrt{F\left(\mathrm{tr}_{\mathcal{M}} V_j \rho_j V_j^\dagger, \mathrm{tr}_{\mathcal{M}} \rho_{j+1}\right)}},$$

ou seja,

$$F\left(\mathrm{tr}_{\mathcal{M}} V_j \rho_j V_j^\dagger, \mathrm{tr}_{\mathcal{M}} \rho_{j+1}\right) = \left(1 - \frac{\eta_j^2}{2}\right)^2.$$

Por fim, sejam $|\psi_1\rangle, \ldots, |\psi_k\rangle$ vetores do espaço $\mathcal{V}\otimes\mathcal{M}\otimes\mathcal{P}$ tais que $\mathrm{tr}_{\mathcal{P}} |\psi_j\rangle\langle\psi_j| = \rho_j$, para $1 \leq j \leq k$.

Como $\mathrm{tr}_{\mathcal{P}} |\psi_j\rangle\langle\psi_j| = \rho_j$ para cada j, é claro que $\mathrm{tr}_{\mathcal{M}\otimes\mathcal{P}} |\psi_j\rangle\langle\psi_j| = \mathrm{tr}_{\mathcal{M}} \rho_j$, e portanto $\mathrm{tr}_{\mathcal{M}\otimes\mathcal{P}} |\gamma_j\rangle\langle\gamma_j| = \mathrm{tr}_{\mathcal{M}\otimes\mathcal{P}} |\psi_{j+1}\rangle\langle\psi_{j+1}|$. Assim, pela proposição 7.5, existe uma transformação unitária Q_{j+1} no espaço $\mathcal{M} \otimes \mathcal{P}$ tal que $(I_{2^v} \otimes Q_{j+1})|\gamma_j\rangle = |\psi_{j+1}\rangle$.

Analogamente, como $\mathrm{tr}_{\mathcal{M}\otimes\mathcal{P}} (V_j \otimes I_{2^p})|\psi_j\rangle\langle\psi_j|(I_{2^p} \otimes V_j^\dagger) = \mathrm{tr}_{\mathcal{M}} V_j \rho_j V_j^\dagger$, então $\mathrm{tr}_{\mathcal{M}\otimes\mathcal{P}} (V_j \otimes I_{2^p})|\psi_j\rangle\langle\psi_j|(I_{2^p} \otimes V_j^\dagger) = \mathrm{tr}_{\mathcal{M}\otimes\mathcal{P}} |\xi_j\rangle\langle\xi_j|$. Logo, pela proposição 7.5, existe uma transformação unitária R_{j+1} no espaço $\mathcal{M} \otimes \mathcal{P}$ tal que $(I_{2^v} \otimes R_{j+1})(V_j \otimes I_{2^p})|\psi_j\rangle = |\xi_j\rangle$.

Podemos definir P_1' de modo que $P_1'|\psi_{init}\rangle = |\psi_1\rangle$, onde $P_1' = I_{2^v} \otimes P_1$ para algum P_1. Em outras palavras, P_1' manipula apenas os qubits que definem o espaço $\mathcal{M} \otimes \mathcal{P}$, e mantém os qubits do verificador (que estão todos zerados). Por fim, seja $P_{j+1} = Q_{j+1}R_{j+1}$ para $1 \leq j \leq k-1$ e $P_{j+1}' = I_{2^v} \otimes P_{j+1}$.

Para $1 \leq j \leq k-1$ temos

$$\begin{aligned}
\left\| P_{j+1}' V_j' |\psi_j\rangle - |\psi_{j+1}\rangle \right\| &= \|(I_{2^v} \otimes Q_{j+1})(I_{2^v} \otimes R_{j+1})(V_j \otimes I_{2^p})|\psi_j\rangle - |\psi_{j+1}\rangle\| \\
&= \|(I_{2^v} \otimes Q_{j+1})|\xi_j\rangle - |\psi_{j+1}\rangle\| \\
&= \|(I_{2^v} \otimes Q_{j+1})|\xi_j\rangle - (I_{2^v} \otimes Q_{j+1})|\gamma_j\rangle\| \\
&= \|(I_{2^v} \otimes Q_{j+1})\left(|\xi_j\rangle - |\gamma_j\rangle\right)\| \\
&= \||\xi_j\rangle - |\gamma_j\rangle\| = \eta_j.
\end{aligned}$$

Desta maneira, utilizando a desigualdade triangular, temos que a norma $\left\|P_k^{'}V_{k-1}^{'}\ldots P_1^{'}|\psi_{init}\rangle - |\psi_k\rangle\right\|$ é limitada superiormente por

$$\begin{aligned}
&\left\|P_k^{'}V_{k-1}^{'}\ldots V_1^{'}P_1^{'}|\psi_{init}\rangle - |\psi_k\rangle\right\| = \left\|P_k^{'}V_{k-1}^{'}\ldots V_1^{'}|\psi_1\rangle - |\psi_k\rangle\right\| \\
&= \left\|P_k^{'}V_{k-1}^{'}\ldots V_1^{'}|\psi_1\rangle - P_k^{'}V_{k-1}^{'}\ldots P_3^{'}V_2^{'}|\psi_2\rangle + P_k^{'}V_{k-1}^{'}\ldots P_3^{'}V_2^{'}|\psi_2\rangle - |\psi_k\rangle\right\| \\
&\leq \left\|P_k^{'}V_{k-1}^{'}\ldots P_3^{'}V_2^{'}(P_2^{'}V_1^{'}|\psi_1\rangle - |\psi_2\rangle)\right\| + \left\|P_k^{'}V_{k-1}^{'}\ldots P_3^{'}V_2^{'}|\psi_2\rangle - |\psi_k\rangle\right\| \\
&= \left\|P_2^{'}V_1^{'}|\psi_1\rangle - |\psi_2\rangle\right\| + \left\|P_k^{'}V_{k-1}^{'}\ldots P_3^{'}V_2^{'}|\psi_2\rangle - |\psi_k\rangle\right\| \\
&= \eta_1 + \left\|P_k^{'}V_{k-1}^{'}\ldots P_3^{'}V_2^{'}|\psi_2\rangle - |\psi_k\rangle\right\| \\
&\vdots \\
&= \sum_{i=1}^{k-2}\eta_i + \left\|P_k^{'}V_{k-1}^{'}|\psi_{k-1}\rangle - |\psi_k\rangle\right\| = \sum_{i=1}^{k-1}\eta_i.
\end{aligned}$$

Observando que $V_k^{'}$ é uma transformação unitária e que transformações unitárias não alteram normas de vetores, podemos concluir que $\left\|P_k^{'}V_{k-1}^{'}\ldots P_2^{'}V_1^{'}|\psi_1\rangle - |\psi_k\rangle\right\| = \left\|V_k^{'}P_k^{'}V_{k-1}^{'}\ldots P_2^{'}V_1^{'}|\psi_1\rangle - V_k^{'}|\psi_k\rangle\right\|$. Além disso, como uma projeção nunca aumenta a norma de um vetor, temos que

$$\left\|\Pi_{acc}^{'}V_k^{'}P_k^{'}V_{k-1}^{'}\ldots P_2^{'}V_1^{'}|\psi_1\rangle - \Pi_{acc}^{'}V_k^{'}|\psi_k\rangle\right\| \leq \sum_{j=1}^{k-1}\eta_j.$$

Como $\rho_k = \left(V_k^{\dagger}\Pi_{acc}V_k\right)\rho_k\left(V_k^{\dagger}\Pi_{acc}V_k\right)$, a superposição $|\psi_k\rangle$ representada parcialmente pela matriz de densidade ρ_k contém apenas estados de aceitação, ou seja, tem $|0\rangle$ no qubit de saída. Do contrário, a aplicação da transformação $V_k^{\dagger}$ após a projeção resultaria numa matriz de densidade distinta de ρ_k. Dessa forma, podemos concluir que $\left\|\Pi_{acc}^{'}V_k^{'}|\psi_k\rangle\right\| = 1$. Portanto, temos que

$$\begin{aligned}
1 &= \left\|\Pi_{acc}^{'}V_k^{'}|\psi_k\rangle\right\| \\
&\leq \left\|\Pi_{acc}^{'}V_k^{'}P_k^{'}\ldots P_2^{'}V_1^{'}|\psi_1\rangle\right\| + \left\|\Pi_{acc}^{'}V_k^{'}|\psi_k\rangle - \Pi_{acc}^{'}V_k^{'}P_k^{'}V_{k-1}^{'}\ldots P_2^{'}V_1^{'}|\psi_1\rangle\right\| \\
&\leq \left\|\Pi_{acc}^{'}V_k^{'}P_k^{'}\ldots P_2^{'}V_1^{'}|\psi_1\rangle\right\| + \sum_{j=1}^{k-1}\eta_j,
\end{aligned}$$

e portanto

$$1 - \sum_{j=1}^{k-1}\eta_j, \leq \left\|\Pi_{acc}^{'}V_k^{'}P_k^{'}\ldots P_2^{'}V_1^{'}P_1^{'}|\psi_{init}\rangle\right\| \leq \sqrt{1-\epsilon}$$

onde a última desigualdade vale pois a probabilidade máxima de aceitação da entrada para qualquer escolha de $P_1, \ldots, P_k$ pelo provador é limitada superiormente por $1-\epsilon$. Assim, $\sum_{j=1}^{k-1} \eta_j \geq 1 - \sqrt{1-\epsilon} \geq \epsilon/2$ (essa última desigualdade vale para todo ϵ).

Pela desigualdade de Cauchy-Schwarz, sabemos que, para duas seqüências $x_1, \ldots, x_k$ e $y_1, \ldots, y_k$ de números reais,

$$\left| \sum_{i=1}^{k} x_i y_i \right| \leq \left(\sum_{i=1}^{k} |x_i|^2 \right)^{1/2} \left(\sum_{i=1}^{k} |y_i|^2 \right)^{1/2}$$

Tomando $x_i = \eta_i$ e $y_i = 1$ para cada i, $1 \leq i \leq k$, concluímos que $\sum_{j=1}^{k-1} \eta_j^2 \geq (\sum_{j=1}^{k-1} \eta_j)^2/k - 1$. Assim sendo, o valor máximo assumido por $\sum_{j=1}^{k-1}(1 - \eta_j^2/2)$ quando $\sum_{j=1}^{k-1} \eta_j \geq \epsilon/2$ é

$$\begin{aligned}
\sum_{j=1}^{k-1} \left(1 - \frac{\eta_j^2}{2}\right) &= k - 1 - \frac{1}{2} \sum_{j=1}^{k-1} \eta_j^2 \\
&\leq k - 1 - \frac{1}{2} \frac{(\sum_{j=1}^{k-1} \eta_j)^2}{(k-1)} \\
&\leq k - 1 - \frac{(\epsilon/2)^2}{2(k-1)} \\
&= k - 1 - \frac{\epsilon^2}{8(k-1)},
\end{aligned}$$

como queríamos. □

Para cada j, $1 \leq j \leq k$, seja ρ_j um elemento de $\mathbf{D}(\mathcal{V} \otimes \mathcal{M})$ que descreve o estado dos registradores $(\mathbf{V}_j, \mathbf{M}_j)$ após a linha 7 do algoritmo do verificador. Como, nas duas primeiras checagens do verificador, a entrada não foi rejeitada, então é verdade que $\rho_1 = \Pi_{init} \rho_1 \Pi_{init}$ e que $\rho_k = (V_k^\dagger \Pi_{acc} V_k) \rho_k (V_k^\dagger \Pi_{acc} V_k)$. Além disso, sabemos por hipótese que o protocolo original aceita a entrada com probabilidade limitada superiormente por $1 - \epsilon$.

Afirmamos que a probabilidade de o verificador aceitar a entrada para cada escolha possível de r é limitada superiormente por

$$p_r = \frac{1}{2} + \frac{1}{2} \sqrt{F\left(\mathrm{tr}_{\mathcal{M}}\, V_r \rho_r V_r^\dagger, \mathrm{tr}_{\mathcal{M}}\, \rho_{r+1}\right)}.$$

Vamos analisar o estado do sistema após o passo 11 do algoritmo do verificador ter sido executado. Tal estado pode ser denotado por $\frac{1}{\sqrt{2}}(|0\rangle|\xi\rangle + |1\rangle|\gamma\rangle)$,

onde $|\xi\rangle$ e $|\gamma\rangle$ são vetores unitários do espaço $\mathcal{V} \otimes \mathcal{K}$. O espaço $\mathcal{V}$ refere-se ao espaço dos qubits de $\mathbf{V}_r$, e $\mathcal{K}$ refere-se aos outros registradores do sistema, a menos de B e V_r. Logo, $\dim(\mathcal{V}) \leq \dim(\mathcal{K})$, e tanto $\mathrm{tr}_{\mathcal{M}}\, V_r \rho_r V_r^\dagger$ como $\mathrm{tr}_{\mathcal{M}}\, \rho_{r+1}$ são elementos de $\mathbf{D}(\mathcal{V})$.

Assim, pela proposição 7.3, temos que $|\langle\xi|\gamma\rangle|^2 \leq F\left(\mathrm{tr}_{\mathcal{M}}\, V_r \rho_r V_r^\dagger, \mathrm{tr}_{\mathcal{M}}\, \rho_{r+1}\right)$, pois $\mathrm{tr}_{\mathcal{K}}\, |\xi\rangle\langle\xi| = \mathrm{tr}_{\mathcal{M}}\, V_r \rho_r V_r^\dagger$ e $\mathrm{tr}_{\mathcal{K}}\, |\gamma\rangle\langle\gamma| = \mathrm{tr}_{\mathcal{M}}\, \rho_{r+1}$. O verificador aplica a transformação de Hadamard em B e aceita a entrada se obtém 0 como resposta.

Logo, a probabilidade de aceitação é dada por

$$\left\| \frac{1}{2}|0\rangle|\xi\rangle + \frac{1}{2}|0\rangle|\gamma\rangle \right\|^2 \;=\; \frac{1}{2} + \frac{1}{2}\mathcal{R}\langle\xi|\gamma\rangle \;\leq\; \frac{1}{2} + \frac{1}{2}\sqrt{|\langle\xi|\gamma\rangle|^2},$$

onde $\mathcal{R}\langle\xi|\gamma\rangle$ indica a parte real de $\langle\xi|\gamma\rangle$. Como $\frac{1}{2} + \frac{1}{2}\sqrt{|\langle\xi|\gamma\rangle|^2}$ é limitado superiormente por $\frac{1}{2} + \frac{1}{2}\sqrt{F\left(\mathrm{tr}_{\mathcal{M}}\, V_r \rho_r V_r^\dagger, \mathrm{tr}_{\mathcal{M}}\, \rho_{r+1}\right)}$, segue a afirmação.

Assim, pelo lema 7.6 e pelo fato de r ter sido escolhido aleatoriamente de maneira uniforme, temos que a probabilidade de o verificador aceitar a entrada é no máximo

$$\begin{aligned}
\sum_{r=1}^{k-1} \frac{p_r}{k-1} &= \sum_{r=1}^{k-1} \left(\frac{1}{k-1}\right) \left(\frac{1}{2} + \frac{1}{2}\sqrt{F\left(\mathrm{tr}_{\mathcal{M}}\, V_r \rho_r V_r^\dagger, \mathrm{tr}_{\mathcal{M}}\, \rho_{r+1}\right)}\right) \\
&\leq \sum_{r=1}^{k-1} \left(\frac{1}{2(k-1)}\right) + \frac{1}{2(k-1)}\left(k - 1 - \frac{\epsilon^2}{8(k-1)}\right) \\
&= \frac{1}{2} + \frac{1}{2} - \frac{\epsilon^2}{16(k-1)^2} \;=\; 1 - \frac{\epsilon^2}{4(m-1)^2}.
\end{aligned}$$

Se o valor de m no protocolo original era par, obtemos a limitação exata descrita no enunciado do teorema. Se m é ímpar, a limitação é ainda mais justa do que a desejada, e portanto segue o resultado. □

Capítulo 8

Comentários Finais

Nesse trabalho, tivemos como principal objetivo estabelecer um paralelo entre o modelo clássico de computação, com origens na década de 30 e o modelo quântico de computação, que teve impulso significativo no começo da década de 80 a partir de alguns estudos de Feynman. Para isso, utilizamos como principal ferramenta os sistemas interativos de prova, que também são bem recentes.

Nos capítulos 2 e 3 da dissertação, tratamos de conceitos e resultados da teoria clássica de complexidade computacional, como máquinas de Turing, classes como **P**, **NP** e **BPP** e a hierarquia polinomial.

No capítulo 4, apresentamos os sistemas interativos de prova e alguns resultados envolvendo tais sistemas, em especial a demonstração de que **PSPACE** = **IP**. A apresentação desse último resultado difere do que é usualmente encontrado na literatura. O estudo desses resultados mostrou a importância do uso de polinômios de grau limitado e do uso de elementos aleatórios na obtenção de resultados não-triviais em teoria de complexidade computacional. O teorema **PCP**, baseado em sistemas interativos de prova, também utiliza fortemente tais ferramentas. Assim, podemos imaginar que a aleatoriedade tem e provavelmente terá por um bom tempo papel de destaque em complexidade computacional.

No capítulo 5 introduzimos o modelo quântico de computação. Definitivamente seria muita pretensão tentar inserir nesse texto uma introdução completa, que abrangesse uma revisão das principais áreas relacionadas ao tema. Optamos por apresentar nesse capítulo os principais conceitos, importantes para a compreensão dos resultados que foram selecionados para fazer parte do texto.

É curioso notar que, no momento, discute-se o modelo quântico de computação principalmente por meio de circuitos, e não com máquinas de Turing. Essa tendência provavelmente permanecerá, visto que a análise do consumo de tempo de algoritmos por meio de máquinas de Turing quânticas é extremamente intrin-

cada e complexa. Por fim, já nesse capítulo mostramos o problema de Deutsch que, apesar de simples, consiste num exemplo que sugere a superioridade do modelo quântico de computação em relação ao clássico.

Nos capítulos 6 e 7, apresentamos os sistemas interativos de prova dentro do modelo quântico de computação.

O estudo dos resultados contidos nessa dissertação mostrou que a área de computação quântica é bastante árdua e intrincada. Muitos detalhes aparentemente irrelevantes nas primeiras leituras mostram-se essenciais depois de muito tempo de estudo, e geralmente nos levam a perceber que o que havia sido entendido num primeiro momento estava totalmente equivocado. A nossa apresentação, em especial dos resultados dos capítulos 6 e 7 é bem mais detalhada do que as que se encontram na literatura. Esperamos que, ao ler esse texto, o leitor tenha menos dificuldades para compreender os resultados expostos do que tivemos ao ler os artigos envolvidos.

Referências Bibliográficas

[AKS02a] M. Agrawal, N. Kayal, e N. Saxena.
Primes is in P (preprint).
`http://www.cse.iitk.ac.in/news/primality.ps`, 2002.

[AKS02b] M. Agrawal, N. Kayal, e N. Saxena.
Primes is in P (revised version).
`http://www.cse.iitk.ac.in/news/primality_v3.ps`, 2002.

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, e M.Szegedy.
Proof verification and hardness of approximation problems.
In *Proc. 33rd Ann. IEEE Symp. on Foundation of Computer Science*, pages 14–23, Los Alamitos, CA, 1992. IEEE Comput. Soc. Press.

[Bab85] L. Babai.
Trading group theory for randomness.
Proc. 17th Ann. ACM Symp. on Theory of Computing, pages 421–429, 1985.

[BBC+95] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N.H. Margolus, P.W. Shor, T. Sleator, J.A. Smolin, e H. Weinfurter.
Elementary gates for quantum computation.
Physical Review A, 52(5):3457–3467, 1995.

[BV97] E. Bernstein e U. Vazirani.
Quantum complexity theory.
SIAM J. Comput., 26(5):1411–1473, 1997.

[CEMM98] R. Cleve, A. Ekert, C. Macchiavello, e M. Mosca.
Quantum algorithms revisited.
R. Soc. Lond. Proc. Ser. A Math. Phys. Eng. Sci., 454(1969):339–354, 1998.

[Chu33] A. Church.
A set of postulates for the foundation of logic.
Annals of Mathematics, 25:839–864, 1933.

[Chu36] A. Church.
An unsolvable problem of elementary number theory.
Annals of Mathematics, 58:345–363, 1936.

[Cob65] A. Cobham.
The intrinsic computational difficulty of functions.
In *Logic, Methodology and Philos. Sci. (Proc. 1964 Internat. Congr.)*, pages 24–30. North-Holland, Amsterdam, 1965.

[Coo71] S.A. Cook.
The complexity of theorem proving procedures.
Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, pages 151–158, 1971.

[Deu85] D. Deutsch.
Quantum theory, the Church-Turing principle and the universal quantum computer.
Proc. Roy. Soc. London Ser. A, 400(1818):97–117, 1985.

[Deu89] D. Deutsch.
Quantum computational networks.
Proc. Roy. Soc. London Ser. A, 425(1868):73–90, 1989.

[Edm65] J. Edmonds.
Paths, trees, and flowers.
Canad. J. Math., 17:449–467, 1965.

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

[GJ79] M. R. Garey e D. S. Johnson.
Computers and intractability - A guide to the theory of NP-completeness.
W. H. Freeman and Co., San Francisco, Calif., 1979.

[GMR85] S. Goldwasser, S. Micali, e C. Rackoff.
The knowledge complexity of interactive proof-systems.
In *Proc. 17th Ann. ACM Symp. on Theory of Computing*, pages 291–304. ACM Press, 1985.

[Gru99] J. Gruska.
Quantum computing.
Advanced Topics in Computer Science Series. McGraw-Hill International (UK) Limited, London, 1999.

[GS86] S. Goldwasser e M. Sipser.

Private coins versus public coins in interactive proof-systems.
In *Proc. 18th Ann. ACM Symp. on Theory of Computing*. ACM Press, 1986.

[Göd31] K. Gödel.
On formally undecidable propositions of Principia Mathematica and related systems.
Monatshefte fur Math. und Physik, 38:173–198, 1931.

[HJW93] L. Hughston, R. Jozsa, e W. Wootters.
A complete classification of quantum ensembles having a given density matrix.
Physics Letters A, 183:14–18, 1993.

[Joz94] R. Jozsa.
Fidelity for mixed quantum states.
Journal of Modern Optics, 41(12):2315–2323, 1994.

[Kar72] R.M. Karp.
Reducibility among combinatorial problems.
Complexity of Computer Computations, pages 85–103, 1972.

[Kle36] S. Kleene.
General recursive functions of natural numbers.
Mathematische Annalen, 112:727–742, 1936.

[Kle52] S. Kleene.
Introduction to Metamathematics.
D. Van Nostrand Co., Inc., New York, N. Y., 1952.

[KS95] Y. Kohayakawa e J.A. Soares.
Demonstrações Transparentes e Impossibilidade de Aproximações.
20º Colóquio Brasileiro de Matemática. IMPA, Rio de Janeiro, 1995.

[KW00] A. Kitaev e J. Watrous.
Parallelization, amplification, and exponential time simulation of quantum interactive quantum systems.
Proc. 32nd ACM Symp. on Theory of Computing, pages 608–617, 2000.

[Lev73] L. A. Levin.
Universal enumeration problems.
Problemy Peredači Informacii, 9(3):115–116, 1973.

[LFKN92] C. Lund, L. Fortnow, H. Karloff, e N. Nisan.
Algebraic methods for interactive proof systems.
Journal of the ACM 39, pages 859–868, 1992.

[Mat70] Y. Matijasevic.
Enumerable sets are diophantine (Russian).
Dokl. Akad. Nauk SSSR, 191:279–282, 1970.
Translation in Soviet Math Doklady, Vol 11, 1970.

[MS73] A. Meyer e L. Stockmeyer.
The equivalence problema for regular expressions with squaring requires exponential time.
Proc. 13th IEEE Symp. on Switching and Automata Theory, pages 125–129, 1973.

[MW04] C. Marriott e J. Watrous.
Quantum Arthur-Merlin games.
Proceedings of the 19th IEEE Annual Conference on Computational Complexity, pages 275–285, 2004.

[NC00] M.A. Nielsen e I.L. Chuang.
Quantum computation and quantum information.
Cambridge University Press, Cambridge, 2000.

[Pap94] C.H. Papadimitriou.
Computational Complexity.
Addison-Wesley Publishing Company, Reading, MA, 1994.

[Pos36] E. Post.
Finite combinatory process.
Journal of Symbolic Logic, 1:103–105, 1936.

[Raz05] R. Raz.
Quantum information and the PCP theorem.
Proceeding of the 46th FOCS, pages 459–468, 2005.

[RSA78] R.L. Rivest, A. Shamir, e L. Adleman.
A method for obtaining digital signatures and public-key cryptosystems.
Comm. ACM, 21(2):120–126, 1978.

[Sha92] A. Shamir.
IP = PSPACE.
J. Assoc. Comput. Mach., 39(4):869–877, 1992.

[Sho94] P.W. Shor.
Algorithms for quantum computation: discrete logarithms and factoring.
In *35th Annual Symposium on Foundations of Computer Science (Santa Fe, NM, 1994)*, pages 124–134. IEEE Comput. Soc. Press, Los Alamitos, CA, 1994.

[Sho97] P.W. Shor.
Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer.
SIAM J. Comput., 26(5):1484–1509, 1997.

[Tur36] A. Turing.
On computable numbers with an application to the entscheidungsproblem.
Proc. London Math. Soc., 2:230–265, 1936.

[Tur37] A. Turing.
Rectifications to 'on computable numbers...'.
In *Proc. London Mathematical Society*, volume 4, pages 544–546, 1937.

[Wat02] J. Watrous.
Limits on the power of quantum statistical zero-knowledge.
Proceedings of IEEE FOCS, pages 495–504, 2002.

[Wat03] J. Watrous.
PSPACE has constant-round quantum interactive proof systems.
Theor. Comput. Sci., 292(3):575–588, 2003.

Índice Remissivo