Provas Holográficas de Tamanho Quase-Linear

Armando Ramos Gouveia

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
UNIVERSIDADE DE SÃO PAULO

DISSERTAÇÃO APRESENTADA PARA
OBTENÇÃO DO GRAU DE MESTRE EM
MATEMÁTICA APLICADA

Área de Concentração: **Ciência da Computação**
Orientador: **Prof. Dr. Yoshiharu Kohayakawa**

Durante a elaboração deste trabalho o autor recebeu apoio financeiro da CAPES.

— São Paulo, novembro de 1998 —

Provas Holográficas de Tamanho Quase-Linear

Armando Ramos Gouveia

Este exemplar corresponde
à redação final da dissertação corrigida,
defendida por Armando Ramos Gouveia
e aprovada pela comissão julgadora.

São Paulo, 13 de novembro de 1998.

Banca examinadora:

- Prof. Dr. Yoshiharu Kohayakawa (orientador) — IME-USP
- Prof. Dr. José Augusto Ramos Soares — IME-USP
- Prof. Dr. Ruy Guerra de Queiroz — DI-UFPe

Aos meus pais

Agradecimentos

Tenho certeza de que há muitas pessoas que contribuíram para a realização desta dissertação e não é possível citá-las todas aqui. No entanto, gostaria de agradecer de uma forma muito especial:

Ao Prof. Yoshiharu Kohayakawa, por sua orientação e pela sua enorme paciência e dedicação de tempo. Esses anos que passamos juntos foram muitíssimo proveitosos para mim.

Ao Prof. Imre Simon, pela orientação de programa, no início do meu mestrado.

Ao colega Claus Akira Matsushigue com o qual aprendi muito sobre este assunto das Provas Checáveis Probabilisticamente.

Ao Jair, ao Orlando e ao Jefferson, pelas várias dicas no uso do LaTeX.

A todos os colegas do IME–USP, junto dos quais vivi milhares de horas de estudo e de agradável convivência. Obrigado pela amizade!

À minha família, a quem devo tudo e mais um pouco.

Abstract

In a system of Probabilistically Checkable Proofs (PCP) the verifier consists of a polynomial time Turing Machine and checks a proof of membership in a given language. This proof is called holographic proof and is given by an oracle, *i.e.*, a computationally unlimited machine. Correct proofs are always accepted and the probability of accepting an incorrect proof is chosen by the verifier and can be as low as desired.

Arora *et al.*, with the famous theorem "$\mathcal{NP} = PCP(\log n, 1)$", showed that it is possible to construct holographic proofs such that the verification is accomplished by reading a constant number of bits from the proof and by using $O(\log n)$ random bits, for inputs of length n.

An improvement on this statement was presented by Polishchuk and Spielman in 1994; they showed another construction that is able to yield a proof of nearly-linear size, which is also checkable in the $PCP(\log n, 1)$ scheme. This dissertation explains what the PCP's are, and shows the construction of those nearly-linear size holographic proofs.

Resumo

Em um sistema de Provas Checáveis Probabilisticamente (PCP), o verificador consiste em uma Máquina de Turing de tempo polinomial, e deve checar uma demonstração de pertinência a uma dada linguagem. Tal demonstração chama-se prova holográfica e é fornecida por oráculo, isto é, uma máquina ilimitada computacionalmente. As provas corretas sempre são aceitas e a probabilidade de se aceitar uma prova incorreta é escolhida pelo verificador e pode ser tão pequena quanto se queira.

Arora *et al.*, com o famoso teorema "$\mathcal{NP} = PCP(\log n, 1)$", mostraram que se pode construir uma prova holográfica cuja verificação se faz com a consulta de um número constante de bits dessa prova e com o uso de $O(\log n)$ bits aleatórios, para entradas de tamanho n.

Uma melhora nesse resultado foi apresentada por Polishchuk e Spielman em 1994, que mostraram outra construção capaz de fornecer uma prova de tamanho quase-linear, a qual também é checável no esquema $PCP(\log n, 1)$. Esta dissertação explica o que são as PCP's e mostra a construção dessas provas holográficas cujo tamanho é quase-linear.

Sumário

Capítulo 1

Introdução

Em Computação utilizam-se vários algoritmos probabilísticos. Na Teoria da Complexidade, o uso desses algoritmos fez surgir, entre outras, a classe IP, Provas Interativas, cujos estudos demonstraram abranger uma classe espetacularmente ampla de problemas.

A área das PCP's, Provas Checáveis Probabilisticamente, abrange métodos que permitem checar a validade de uma prova examinando-se apenas uma quantidade pequena de seus bits. Se um provador escreve uma prova em certo formato — prova holográfica — e afirma que essa é a prova de um teorema, existe um algoritmo pelo qual nós podemos escolher aleatoriamente uns poucos bits da prova para ler e, após fazer alguns testes simples sobre esses bits, decidir se aceitamos ou rejeitamos a prova. Uma prova correta sempre será aceita. Uma prova falsa será rejeitada com probabilidade maior que $1/2$. Podemos repetir esse algoritmo algumas vezes para obter um grau de certeza arbitrariamente grande de que esse teorema é verdadeiro.

O resultado principal dessa nova área surge em [ALM+92], mostrando que, para qualquer teorema (i.é, instância aceita) de uma linguagem da classe $\mathcal{NP}$, existe uma prova holográfica que pode ser verificada pela consulta de uma quantidade *constante* de bits, e usando apenas um número de bits aleatórios que é logarítmico no tamanho da instância.

Esse resultado é apresentado no Capítulo 3 desta dissertação, juntamente com as principais noções sobre as demonstrações probabilísticas.

Uma significativa melhoria do resultado de [ALM+92] foi dada por Polishchuk e Spielman [PS94] que apresentaram outra demonstração do mesmo resultado — construção de provas checáveis pela consulta de $O(1)$ bits e com uso de $O(\log n)$ bits aleatórios — mas com uma diminuição no tamanho da prova holográfica construída, de polinomial para quase-linear. Ou seja, para um dado teorema, qualquer prova convencional de tamanho N pode ser convertida em uma prova holográfica de tamanho

$N^{1+\varepsilon}$, para todo $\varepsilon > 0$. O principal objetivo deste trabalho é explicar a demonstração desse teorema de [PS94].

Tal demonstração requer conhecimentos de várias áreas da Matemática. Por isso foi incluído o Capítulo 2, com alguns conceitos que utilizamos de Álgebra, Lógica e Teoria dos Códigos, além de uma noção geral de Teoria da Complexidade, necessária para entender o alcance dos assuntos tratados.

Para chegar ao nosso teorema, em primeiro lugar, reduzimos o problema $\mathcal{NP}$-completo de Satisfazibilidade de Circuitos Booleanos para um problema de coloração de grafos tal que o grafo é colorível com determinadas regras se e somente se o circuito original for satisfazível. Essa redução é descrita no Capítulo 5.

No Capítulo 6, é dada uma descrição algébrica desse grafo, que permite identificar certos nós do grafo com elementos de $\mathrm{GF}(2^n)$ de tal modo que os nomes dos vizinhos de um nó podem ser expressos como polinômios de grau baixo no nome desse nó. Em seguida, baseando-se em polinômios, faz-se uma "aritmetização" das regras de coloração.

Com os polinômios, chegamos a um ponto em comum com o artigo [ALM+92] e outros trabalhos dessa área. O Capítulo 4 fornece resultados obtidos por vários autores, que servirão como ferramentas para lidar com o nosso problema aritmetizado.

O Capítulo 7 contém a construção de uma prova holográfica para esse problema, e a demonstração de que essa prova pode ser verificada com a consulta de um número constante de segmentos; porém o tamanho de cada segmento depende do tamanho da instância do problema. A prova final é obtida no Capítulo 8, mediante um processo recursivo de composição das provas holográficas.

As principais contribuições deste trabalho encontram-se nos Capítulos 7 e 8, em que se buscou explicar aspectos cujos detalhes não se encontram nos textos originais. Os Capítulos 4 a 6 seguem de perto [Spi95].

Este trabalho não pretende apresentar todos os conceitos básicos e supõe que o leitor já tenha certa familiariedade com esta área. Por isso, antes de iniciar a leitura desta dissertação, o leitor interessado pode consultar [KS95] e [Bab94] para uma introdução a este recente ramo da Ciência da Computação.

Capítulo 2

Conceitos Básicos

Neste capítulo veremos algumas noções gerais sobre a Teoria da Complexidade e sobre mais alguns ramos da Matemática. Também veremos certas notações e nomenclaturas que nos serão muito úteis.

2.1 Em geral

Em Ciências Exatas, um tradicional descuido é a não tradução dos termos estrangeiros ou o seu uso sem indicação em itálico. Esta dissertação também pretende seguir a linha tradicional e, para que a coisa se torne legítima, veremos alguns esclarecimentos.

Palavras como *checar* e *checagem*, *randômico* e *randomização*, já aparecem no dicionário[1]. O significado de *bit* também aparece no dicionário, mas com indicação de que esse vocábulo pertence à língua inglesa. Quanto a *string*, que significa "cadeia de caracteres", não será traduzida pois já é um termo bastante conhecido em Computação.

A partir deste ponto, as palavras citadas e suas derivadas serão utilizadas *sem* itálico.

2.2 Sobre Teoria da Complexidade

Vejamos, informalmente, alguns resultados e definições utilizados neste trabalho. São definidas apenas as notações que mais interessam, supondo um conhecimento básico do assunto (notação O, máquina de Turing, ...). Maiores detalhes, e os aspectos formais, podem ser encontrados em [Pap94] e [KS95].

Neste trabalho, todas as referências a complexidade de tempo dizem respeito ao tempo determinístico.

[1] Aurélio Buarque de Holanda, Ed. Nova Fronteira, 2ª edição, 1995.

Denotamos as funções polinomiais por $n^{O(1)}$, ou seja, $f(n) = n^{O(1)}$ se e somente se existe uma constante k tal que $f(n) = O(n^k)$.

Uma função f é *polilogarítmica* quando é a composta de uma polinomial com um logarítmica, ou seja, $f(n) = (\log n)^{O(1)}$. A notação $\log^{O(1)} n$ também é utilizada.

Dizemos que uma função f é *quase-linear* se, para qualquer $\varepsilon > 0$, vale que $f(n) = O(n^{1+\varepsilon})$.

Não é difícil constatar que, para quaisquer constantes positivas ε e k, vale $n^{\varepsilon} > (\log n)^k$ para $n > N_0$ suficientemente grande. Portanto, qualquer função g que seja "linear vezes polilogarítmica" também será quase-linear, pois

$$g(n) = n(\log n)^{O(1)} = O(n^{1+\varepsilon}),$$

para qualquer $\varepsilon > 0$.

Recordemos algumas coisas sobre os problemas de decisão, isto é, aqueles cuja resposta é SIM ou NÃO. Um fato bastante conhecido é que decidir sobre a resposta a um problema de decisão equivale a decidir sobre a pertinência a uma linguagem.

Definição 2.1 *A classe $\mathcal{P}$ compreende as linguagens para as quais a pertinência pode ser decidida em tempo polinomial.*

Definição 2.2 *A classe $\mathcal{NP}$ é a classe das linguagens L tais que, para todo $x \in L$, existe uma prova y de que $x \in L$; e existe algoritmo de tempo polinomial no $|x|$, para verificar a prova.*

Na definição acima, o *problema* é determinar se x pertence a L; y é chamada *solução* do problema ou *certificado* de que $x \in L$; x é uma *instância* do problema; e L é uma *$\mathcal{NP}$-linguagem.*

Portanto, $\mathcal{NP}$ é a classe das linguagens para as quais os certificados de pertinência podem ser verificados em tempo polinomial.

A classe "complementar" à $\mathcal{NP}$ é chamada *co$\mathcal{NP}$* e é definida substituindo "$\in$" por "$\notin$" na definição anterior. Em outras palavras, *co$\mathcal{NP}$* é a classe das linguagens para as quais existe algoritmo polinomial para verificar os certificados de *não-pertinência.*

Definição 2.3 PSPACE *é a classe das linguagens para as quais a pertinência pode ser decidida em espaço polinomial. Não há limitação de tempo.*

Na Teoria da Complexidade, há questões básicas em aberto. Por exemplo, sabe-se que PSPACE contém $\mathcal{NP}$, *co$\mathcal{NP}$* e várias outras classes; no entanto ainda não foi provada sequer a conjectura "$\mathcal{P} \neq$ PSPACE".

Mas o maior problema em aberto de toda a Teoria da Computação é

$$\mathcal{P} \neq \mathcal{NP} \ ?$$

A conjectura mais aceita é que realmente sejam diferentes. Outra questão em aberto é

$$\mathcal{NP} \neq co\mathcal{NP} \ ?$$

Se forem diferentes, então também $\mathcal{P} \neq \mathcal{NP}$; mas, se forem iguais, isso não implica que $\mathcal{P} = \mathcal{NP}$.

E é esse contexto — das questões em aberto — que talvez seja transformado com o novo ramo da Complexidade envolvendo as Provas Checáveis Probabilisticamente (PCP's), cujo poder foi estudado entre os anos 1989 e 1992 por diversos autores. Tais estudos proporcionaram uma rápida sucessão de artigos com demonstrações e refinamentos de resultados, culminando no Teorema 3.7, que dá uma nova (e surpreendente) caracterização de $\mathcal{NP}$. Veremos esse teorema no Capítulo 3, juntamente com as explicações sobre o que são as PCP's.

2.3 Sobre Álgebra

Neste trabalho lidamos com corpos finitos. Para todo primo p e todo inteiro n, existe um único corpo finito de ordem p^n (ou seja, um corpo com p^n elementos), o qual é chamado GF(p^n), onde GF vem de "Galois Field".

O corpo GF(2) possui os elementos $\{0, 1\}$ com a multiplicação normal e com a adição igual à "soma módulo 2". Nos Capítulos 6 em diante, trabalhamos com o corpo GF(2^n), para n inteiro. Repare que GF(2^n) não é o mesmo que [GF(2)]^n.

Um *gerador* de GF(2^n) é um elemento α tal que $\alpha^{2^n-1} = 1$ e $\alpha^k \neq 1$ para todo k tal que $0 < k < 2^n - 1$. Todo corpo finito possui um gerador pois o grupo multiplicativo formado por todos os elementos não-nulos desse corpo finito é cíclico.

Dado um gerador α para GF(2^n), cada elemento desse corpo pode ser representado por um polinômio de grau menor que n em α com coeficientes em $\{0, 1\}$.

2.4 Sobre Lógica

Um problema do nosso interesse é o CircSat — Satisfazibilidade de Circuitos.

Definição 2.4 Problema CircSat

Instância: *um circuito C, composto de portas NOT e de portas binárias AND e OR.*

Pergunta: *existe alguma atribuição de bits para as entradas de C que faça esse circuito calcular saída igual a 1?*

Tratam-se de circuitos normais, onde cada bit de entrada pode ser usado em várias portas. Além disso, uma mesma saída de uma porta pode servir de entrada para várias outras (i.é, o *fan-out* não é limitado). Como única restrição temos *fan-in* $= 2$, ou seja, as portas AND e OR possuem dois bits de entrada.

Para mostrar que um problema π é $\mathcal{NP}$-completo, o método tradicional é mostrar que o SAT — ou qualquer outro problema $\mathcal{NP}$-completo — pode ser polinomialmente reduzido a π. O SAT consiste em, dada uma expressão booleana ϕ na forma normal conjuntiva, decidir se ϕ é satisfazível. E é relativamente fácil reduzir o SAT ao CircSat, pois tanto as expressões como os circuitos são maneiras de representar funções booleanas.

Outra interessante demonstração de que o CircSat é $\mathcal{NP}$-completo pode ser encontrada em [Pap94], onde, sem passar pela redução ao problema SAT, Papadimitriou prova diretamente que qualquer linguagem da classe $\mathcal{NP}$ pode ser reduzida ao CircSat.

2.5 Sobre Teoria dos Códigos

Vejamos alguns conceitos sobre códigos em geral. No fim desta seção falaremos sobre um tipo especial chamado código de correção-de-erro.

Seja Σ um alfabeto com q letras. Um *código* $\mathcal{C}$ de tamanho n sobre Σ é um subconjunto de Σ^n. Um elemento $c \in \mathcal{C}$ é chamado *palavra-código*. A menos que algo diferente seja especificado, os códigos citados neste trabalho serão códigos sobre o alfabeto $\{0, 1\}$, chamados *códigos binários*.

Dois importantes parâmetros de um código são sua taxa e sua distância mínima. A *taxa* de um código $\mathcal{C}$ é $(\log_q |\mathcal{C}|)/n$; isso indica quanta informação, em média, está contida em cada símbolo. A *distância mínima* de um código $\mathcal{C}$ é $\min_{x,y \in \mathcal{C}; x \neq y} d(x, y)$, onde $d(x, y)$ é a distância de Hamming entre duas palavras (i.é, o número de posições em que elas diferem). Falaremos bastante sobre

$$\frac{1}{n} \min\{d(x, y) : x, y \in \mathcal{C};\ x \neq y\},$$

que é a *distância mínima relativa* de um código.

Como estamos interessados no desempenho assintótico dos códigos, definimos uma *família de códigos* como sendo uma seqüência infinita de códigos que contém no máximo um código de cada tamanho. Se $\{\mathcal{C}_i\}$ é uma família de códigos tais que, para todo i, a taxa de $\mathcal{C}_i$ é maior que r, então dizemos que a família tem taxa pelo menos r.

Similarmente, se a distância mínima relativa de cada código na família é pelo menos δ, então dizemos que a distância mínima relativa da família é pelo menos δ.

Uma família de códigos sobre um alfabeto fixo é chamada *assintoticamente boa* se existem constantes positivas r e δ tais que a família tem taxa e distância mínima relativa pelo menos r e δ respectivamente.

Neste trabalho lidamos com dois tipos de código:

(i) códigos polinomiais, que vemos no Cap. 4 e são codificações de polinômios de várias variáveis;

(ii) códigos de correção-de-erro, que veremos a seguir, e são codificações de strings (de 0's e 1's) genéricas.

Os códigos de correção-de-erro (*error-correcting codes*) surgiram para lidar com um problema fundamental na comunicação: quando uma mensagem é enviada de um lugar a outro, freqüentemente é distorcida ao longo do caminho. Um código de correção-de-erro fornece um modo sistemático de acrescentar informação a uma mensagem para que, mesmo que uma parte dessa mensagem seja corrompida na transmissão, o receptor consiga, apesar disso, deduzir o que o autor da mensagem pretendeu transmitir. Naturalmente, a probabilidade do receptor conseguir recuperar a mensagem original diminui conforme cresce a quantidade de distorção. Similarmente, a quantidade de distorção que o receptor consegue tolerar aumenta quanto mais informação redundante for adicionada à mensagem transmitida.

Intuitivamente, um bom código de correção-de-erro é um enorme conjunto de palavras tais que cada duas palavras do conjunto diferem entre si em muitas posições.

Um problema central da Teoria dos Códigos é encontrar construções explícitas de famílias de códigos de correção-de-erro assintoticamente boas com taxa e distância mínima relativa tão grandes quanto possível.

Spielman, nos Capítulos 2 e 3 de [Spi95], constrói códigos chamados *superconcentradores*, que formam famílias de códigos de correção-de-erro assintoticamente boas. Essa é a primeira construção conhecida de tais códigos que possui algoritmos de tempo linear (i.é, extremamente eficientes) para codificação e também para decodificação.

Capítulo 3

Provas Checáveis Probabilisticamente

O resultado em estudo neste trabalho insere-se na recente área da Computação chamada PCP. Neste capítulo veremos uma explanação geral sobre o contexto desse resultado, bem como uma introdução às Provas Checáveis Probabilisticamente.

3.1 Demonstrações probabilísticas

Vários algoritmos probabilísticos são utilizados em Computação. Por exemplo, na área de Algoritmos, surgiram os algoritmos Monte Carlo que dão respostas rápidas e quase 100% corretas para questões extremamente difíceis de computar. Também há as interações de "conhecimento zero" em que alguém possui uma informação sigilosa e é capaz de, sem revelar o segredo, demonstrar possuí-lo. Em Criptografia há aplicações práticas que utilizam tais sistemas.

Nos algoritmos probabilísticos que veremos, não é lida a entrada inteira, mas escolhem-se, por sorteio, certas partes a serem examinadas. Após essa escolha, está terminado o aspecto aleatório e, a partir de então, o algoritmo funciona de modo perfeitamente determinístico sobre esses dados. No caso de um algoritmo com múltiplas interações entre provador e checador, cada interação é dividida nesses dois estágios (primeiro, escolha aleatória de dados da entrada; segundo, execução determinística de um algoritmo sobre esses dados escolhidos). E é assim que esse sistema obtido, visto como um todo, funciona de forma probabilística.

Como vimos na Seção 2.2, cada problema de decisão da classe $\mathcal{NP}$ pode ser interpretado como um problema de decidir sobre a pertinência a uma certa linguagem. Para a definição seguinte, veja aspectos formais em [GMR85].

Definição 3.1 *Um* sistema interativo de provas *para uma linguagem L, é um jogo entre duas partes:*

(i) um verificador V, limitado polinomialmente e com acesso a bits aleatórios;

(ii) um provador P, ilimitado computacionalmente (dado por oráculo).

O funcionamento do sistema consiste em que o verificador, baseado numa instância x, executa uma estratégia e, consultando bits aleatórios, interage com o provador. O objetivo de P é convencer V de que $x \in L$. Se isto acontecer dizemos que P ganhou *o jogo; caso contrário, dizemos que P* perdeu *o jogo. O sistema deve satisfazer:*

(a) se $x \in L$, então P sempre possui estratégia ganhadora;

(b) se $x \notin L$, então qualquer estratégia escolhida por P é derrotada com probabilidade $\geq 1/2$.

Definição 3.2 IP *é a classe das linguagens para as quais a pertinência possui um sistema interativo de provas.*

Vários teoremas surgiram na tentativa de estabelecer o alcance desses tipos de demonstração. A seguir temos o resultado definitivo, que mostra quão poderosa é IP, e que causou entusiasmo nessa área da Computação.

Teorema 3.3 *$IP = PSPACE$.*

A demonstração desse resultado foi feita em [Sha90], baseada em trabalho anterior de [LFKN90]. Uma demonstração do entusiasmo pode ser encontrada em [Bab90].

Diante da conjectura amplamente aceita de que a classe $\mathcal{NP}$ esteja contida *propriamente* em PSPACE, o teorema acima significa que as provas interativas são aplicáveis a muitos mais problemas além daqueles que se encontram na classe $\mathcal{NP}$.

Há diversos tipos de sistemas interativos; alguns são de interesse apenas teórico, e outros possuem aplicação bastante prática, com algoritmos desenvolvidos para a área de Criptografia. Cada sistema faz parte de alguma sub-classe de complexidade dentro da classe IP. Vários autores estudaram a eficiência de tais sistemas e suas abrangências em relação aos problemas das conhecidas classes $\mathcal{P}$, $\mathcal{NP}$ etc. As definições dos particulares sistemas interativos podem ser encontradas em [Gol94], onde Goldreich enuncia também os teoremas principais envolvendo cada um.

3.2 O esquema PCP

Entre as demonstrações probabilísticas, há um tipo especial, em que a regra não permite interação entre provador e checador após o sorteio de bits aleatórios. Ou seja, o provador deve apresentar de antemão todos os dados que o checador possa precisar consultar para fazer a verificação.

Definição 3.4 *Um sistema de provas checáveis probabilisticamente* (sistema pcp) *para uma linguagem L é constituído de um verificador V limitado polinomialmente; e de um oráculo que, para cada instância x, fornece um certificado π no qual V se baseia para aceitar ou rejeitar essa instância. E o sistema é tal que*

(i) se $x \in L$, existe certificado π_x tal que V aceita (x, π_x) com probabilidade 1;

(ii) se $x \notin L$, então, para qualquer π, o verificador V rejeita (x, π) com probabilidade $\geq 1/2$.

O certificado π acima é chamado de *prova holográfica* (ou *prova transparente*); e os sistemas pcp também são chamados de *sistemas de provas holográficas.*

As probabilidades provêm dos bits aleatórios acessados por V. Uma importante constatação é que, se $x \notin L$, então, fazendo k repetições da verificação, a probabilidade de x não ser rejeitado é no máximo 2^{-k}. É dessa forma que se obtém, para qualquer $\varepsilon > 0$, uma verificação que rejeita os certificados falsos com probabilidade pelo menos $1 - \varepsilon$. Por exemplo, para $k = 10$, os certificados falsos são rejeitados com probabilidade superior a 99,9%.

A grande vantagem de um sistema probabilístico desse tipo é permitir que o verificador, *em tempo razoável*, faça a checagem de uma prova π (previamente apresentada). A idéia é que V não precisa "olhar" para a demonstração inteira. Em vez disso, com acesso a bits aleatórios, sorteiam-se *algumas* posições de π nas quais será feita a verificação, obtendo-se uma comprovação conforme a Definição 3.4.

Portanto, a eficiência é medida pelo número de consultas feitas à prova e pelo número de bits aleatórios usados (para sortear as posições a serem consultadas). Com base nesses dois parâmetros de mais interesse, classificam-se os sistemas nas classes de complexidade definidas a seguir.

Definição 3.5 *A classe de complexidade $PCP(r(\cdot), q(\cdot))$ é o conjunto das linguagens para as quais a pertinência possui um sistema pcp no qual V, para qualquer instância de tamanho n, utiliza no máximo $O(r(n))$ bits aleatórios e faz no máximo $O(q(n))$ consultas ao certificado.*

Notação 3.6 *Para conjuntos R e Q de funções, denotaremos*

$$PCP(R,Q) = \bigcup_{r\in R,\ q\in Q} PCP(r,q).$$

A fim de não usar o mesmo nome para coisas desiguais, vamos tentar diferenciar as notações. Estamos usando "pcp" para o sistema de provas da Def. 3.4; "$PCP(\cdot,\cdot)$" para as classes da Def. 3.5; e "PCP" para o assunto — dentro Ciência da Computação — que estamos estudando.

O poder das pcp's foi estudado em uma seqüência de grandes trabalhos, que culminaram no seguinte resultado, de Arora, Lund, Motwani, Sudan e Szegedy [ALM+92], que é bastante surpreendente.

Teorema 3.7 $\mathcal{NP} = PCP\ (\log n,\ 1)$.

Ou seja, qualquer linguagem da classe $\mathcal{NP}$ possui um sistema de checagem que, para verificar um certificado de pertinência, lê apenas um número constante de caracteres, os quais são escolhidos com base em apenas uma quantidade de bits aleatórios que é logarítmica no tamanho da entrada.

Tal resultado, longe de ser a confirmação de algo intuitivo, foi totalmente inesperado na época. Afinal, ele afirma que o número de bits lidos pelo checador é uma constante, independente do tamanho da instância.

Uma boa exposição da demonstração completa desse teorema pode ser encontrada em [HPS94]. A idéia geral da prova é apresentar uma pcp para o problema 3SAT, que é $\mathcal{NP}$-completo. Para isso, são feitas codificações aritméticas das fórmulas booleanas e também das soluções (atribuições que satisfazem a fórmula em questão) apresentadas pelo oráculo.

Tais codificações são feitas através de polinômios. Grosseiramente falando, isso permite que possam ser checadas probabilisticamente examinando-as em poucas posições, pois dois polinômios de grau baixo podem coincidir apenas em uma fração pequena do seu domínio (desde que esse domínio tenha tamanho razoável).

Várias ferramentas são utilizadas para lidar com esses polinômios, incluindo os conhecidos "testes de grau baixo". Matsushigue, em [Mat97], além da parte computacional da pcp para o 3SAT, apresenta também o intrincado arcabouço técnico sobre a parte algébrico-probabilística necessária para chegar aos teoremas envolvendo tais polinômios de grau baixo.

Uma das maiores motivações para a pesquisa na área das PCP's foi a sua conexão com a área de Otimização, de onde já surgiram importantes repercussões práticas. Junto

com a demonstração de que $\mathcal{NP} = PCP(\log n, 1)$, o próprio artigo [ALM+92] também mostra que, como conseqüência, obtêm-se provas de resultados de não-aproximabilidade para muitos problemas.

As explicações dessa e de outras afirmações derivadas dos sistemas probabilísticos podem ser encontradas em [Bab94]. Tais resultados significam que, se $\mathcal{P} \neq \mathcal{NP}$ então, para certas classes de problemas, não se pode encontrar (em tempo polinomial) nem sequer uma solução aproximada.

Voltemos a considerar o Teorema 3.7, desta vez olhando para algum problema fixo pertencente à classe $\mathcal{NP}$. Tomemos, por exemplo, o CircSat (veja Def. 2.4). Chegamos à seguinte conclusão.

Corolário 3.8 *Para o problema CircSat, existe um verificador V de tempo polinomial que recebe como entrada um circuito C e uma prova holográfica π, e comporta-se do seguinte modo.*

(i) *V lê apenas um número constante de bits de π;*

(ii) *V usa $O(\log |C|)$ bits aleatórios;*

(iii) *se C é satisfazível, então existe π_C tal que V aceita (C, π_C) com probabilidade 1;*

(iv) *se C não é satisfazível, então, para qualquer π, o verificador V rejeita (C, π) com probabilidade $\geq 1/2$;*

(v) *o certificado π tem tamanho $O(|C|^3)$.*

O item (v) acima decorre de que, pela construção de [ALM+92], a prova holográfica tem tamanho $O(n^3)$, onde n é o tamanho da instância. A demonstração do Teorema 3.7 também aparece em [Sud92] e sua construção fornece provas de tamanho $O(n^2)$, ainda polinomial. Polishchuk e Spielman, em [PS94], melhoraram ainda mais o resultado e apresentaram construções que, para qualquer $\varepsilon > 0$, obtêm provas holográficas de tamanho $n^{1+\varepsilon}$. A meta dos próximos capítulos é justamente mostrar essa construção de tamanho quase-linear.

Na Seção 8.8 aparece uma discussão mais detalhada sobre o tamanho das provas obtidas para problemas genéricos na classe $\mathcal{NP}$.

Capítulo 4

Códigos Polinomiais

O principal elemento da nossa construção de provas holográficas é o modo pelo qual um polinômio de três variáveis será codificado. Antes de chegar a eles, vamos primeiro passar pelos polinômios de duas variáveis, e veremos resultados sobre suas codificações.

4.1 Códigos checáveis e verificáveis

Nesta seção definimos checadores e verificadores e apresentamos a terminologia que será utilizada para fazer suas análises.

Definição 4.1 *Um* checador *para uma família de códigos $\{C_i\}$ de tamanhos $\{n_i\}$ e distância mínima relativa δ sobre o alfabeto Σ é um algoritmo probabilístico tal que*

- *o checador aceita cada palavra de C_i com probabilidade 1;*
- *se a probabilidade de o checador aceitar a palavra w de tamanho n_i é maior que $1/2$, então existe uma (única) palavra-código de C_i de distância mínima relativa no máximo $\delta/3$ em relação a w.*

Medimos a eficiência de um checador pelo número de bits que ele lê da entrada.

Vamos impor mais estrutura aos bits de uma palavra-código e aos modos pelos quais o checador pode acessá-los. Particionamos os bits do código em *segmentos* e obrigamos o checador a ler todos os bits do segmento se ele quiser ler qualquer um. Por exemplo, se s é um segmento que contém do i-ésimo até o j-ésimo bit, e se $\vec{x} = (x_1, x_2, \ldots, x_n)$ é uma palavra, então o valor de x no segmento s é $(x_i, \ldots, x_j)$. Se um algoritmo lê x_i, vamos mudá-lo para que também leia $x_{i+1}, \ldots, x_j$, ou seja, penalizamos um pouco a eficiência para ganharmos certa estrutura. Note que definimos segmentos como conjuntos *disjuntos* de posições. Quando descrevemos um checador de um código cujos bits estão divididos

em segmentos, contamos quantos segmentos do código o checador lê e quantos bits esse segmento contém.

É bom destacar o fato de que essa divisão dos bits em segmentos pode existir apenas na mente do checador. Ela não implica que qualquer bit de formatação apareça nos dados. Descrevemos os bits de um código como estando divididos em segmentos para indicar o modo pelo qual os nossos algoritmos vão acessar os bits desse código.

Além de ser checável, um código *verificável* tem a propriedade de que é possível verificar probabilisticamente se foram corrompidos os bits individuais de uma palavra recebida.

Definição 4.2 *Seja $\{C_i\}$ uma família de códigos de tamanhos n_i e distância mínima relativa δ sobre o alfabeto Σ tal que, em cada código, os bits das palavras são particionados em segmentos. Um* verificador *de $\{C_i\}$ é um algoritmo probabilístico que recebe uma palavra w e o nome de um segmento s como entrada tal que*

- *se w é uma palavra de C_i, o verificador aceita;*
- *se a probabilidade de que o verificador aceite uma palavra w (de tamanho n_i) no segmento s é maior que $1/2$, então existe uma (única) palavra-código c de C_i de distância mínima relativa no máximo $\delta/3$ em relação a w tal que c tem o mesmo valor que w no segmento s.*

A eficiência de um verificador é medida do mesmo modo que a do checador: número de segmentos lidos e número de bits contidos em cada segmento.

4.2 Polinômios de duas variáveis

Vamos construir codificações de polinômios de duas variáveis e de um certo grau. Após isso veremos os teoremas demonstrando que tais codificações possuem os checadores e verificadores que desejamos.

Essas codificações são construídas sobre um corpo. Tal corpo será denotado por $\mathcal{G}$ e não faz diferença qual seja, desde que possua tamanho suficientemente grande para conter os objetos que descrevemos.

Definição 4.3 *Um polinômio $p(x_1, \ldots, x_k)$ tem grau $(d_1, \ldots, d_k)$ se o grau de p em x_i é no máximo d_i, para cada i.*

Há várias maneiras de exibir um polinômio de grau (d, e). A mais natural é escrever seus coeficientes, como na Figura 4.1(a). Uma outra maneira de exibir tal polinômio é

escolher conjuntos $X \subseteq \mathcal{G}$ e $Y \subseteq \mathcal{G}$ tais que $|X| > d$ e $|Y| > e$, e então listar os valores de p em cada ponto de $X \times Y$, como na Figura 4.1(b). É fácil verificar que, assim, o polinômio p está definido univocamente. Ambas essas maneiras exigem que se escreva pelo menos $(d+1)(e+1)$ unidades de informação, i.é, elementos de $\mathcal{G}$.

Nossa maneira de exibir polinômios vai conter ainda mais informação. Embora os dados extras sejam desnecessários para especificar o polinômio, eles serão úteis quando quisermos estabelecer fatos sobre o polinômio sem ler a sua descrição inteira.

Agora definimos uma *apresentação* de um polinômio como sendo a lista dos seus valores sobre algum domínio, juntamente com a lista de polinômios de uma variável obtidos quando uma das variáveis é restrita a um valor no domínio. Veja Figura 4.1(c).

Definição 4.4 *Seja $p(x, y)$ um polinômio de grau (d, e) sobre um corpo $\mathcal{G}$ e sejam X, $Y \subseteq \mathcal{G}$. Uma* apresentação correta *de p sobre $X \times Y$ consiste em*

- *a lista dos valores de p em cada ponto de $X \times Y$;*
- *para cada $x_0 \in X$, os coeficientes do polinômio de uma variável obtido pela restrição de p a $\{x_0\} \times \mathcal{G}$;*
- *para cada $y_0 \in Y$, os coeficientes do polinômio de uma variável obtido pela restrição de p a $\mathcal{G} \times \{y_0\}$.*

Quanto à estruturação em segmentos comentada na pág. 15, as apresentações seguirão a seguinte divisão:

- cada valor de p listado na apresentação é um segmento separado;
- a descrição inteira (i.é, o conjunto de coeficientes) de cada polinômio de uma variável constitui um único segmento.

Quando falamos "lista dos valores de p", queremos dizer que os dados devem estar organizados em uma lista na qual cada elemento possui o mesmo tamanho, de modo que, se alguém quiser consultar o valor de p em (x, y), saberá instantaneamente o lugar da lista em que deve olhar. O mesmo precisa valer para cada lista de polinômios de uma variável.

Uma vez definidas as apresentações corretas, relembremos o nosso objetivo de montar demonstrações probabilísticas, conforme os conceitos do capítulo anterior. O checador vai pedir a codificação de certo polinômio e, como resposta, o oráculo vai exibir alguns dados. Esses dados podem ser verdadeiros ou falsos, e é por isso que o checador deve executar certos algoritmos de verificação.

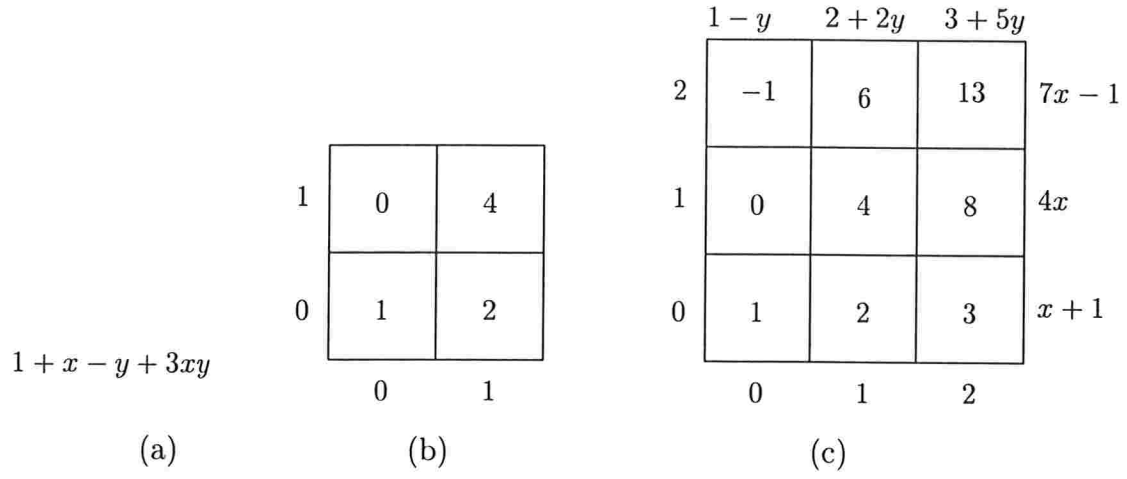

Figura 4.1: Três maneiras de descrever um polinômio

Tais dados serão chamados simplesmente de *apresentação* e podem constituir ou não uma apresentação correta. Como o oráculo pode exibir informações falsas, então, em princípio, o conteúdo de uma apresentação pode ser qualquer coisa; portanto a melhor descrição que podemos dar é dizer que se trata de uma string que é do mesmo tamanho que uma apresentação correta.

Definição 4.5 *Uma* (d, e)-apresentação *sobre* $X \times Y$ *consiste em*

- *uma string com o mesmo tamanho que teria uma lista com um elemento de* $\mathcal{G}$ *para cada ponto de* $X \times Y$. *Dizemos que essa string* atribui *um elemento de* $\mathcal{G}$ *para cada ponto de* $X \times Y$;
- *para cada* $x_0 \in X$, *uma string com o mesmo tamanho da descrição de um polinômio de uma variável de grau* e *sobre* $\mathcal{G}$;
- *para cada* $y_0 \in Y$, *uma string com o mesmo tamanho da descrição de um polinômio de uma variável de grau* d *sobre* $\mathcal{G}$.

Quando o grau (d, e) *estiver claro a partir do contexto, escreveremos apenas* apresentação.

Daqui por diante, o que for dito genericamente sobre apresentações aplica-se também às apresentações corretas (Def. 4.4), que podem ser vistas como um caso particular de apresentação.

Ao conjunto $X \times Y$ na definição anterior chamamos *domínio* da apresentação.

A divisão da string em segmentos é apenas por conveniência da descrição, e indica o modo como estes segmentos serão interpretados pelos algoritmos que os lêem.

Agora nos interessam resultados que auxiliem na tarefa de descobrir se uma apresentação é ou não correta. Será essencial a característica de que dois polinômios distintos coincidem apenas em um número pequeno de pontos.

Lema 4.6 *Seja p um polinômio de grau (d, e) sobre um domínio $X \times Y$. Se p tem mais do que*

$$|X| \cdot |Y| \cdot \left(\frac{d}{|X|} + \frac{e}{|Y|} \right)$$

raízes em seu domínio, então p é o polinômio nulo.

Prova. Feita por Schwartz, em [Sch80]. □

Nossa meta é o seguinte resultado. Se recebemos uma apresentação e nos dizem que ela corresponde a um polinômio de duas variáveis, podemos checar isso probabilisticamente lendo apenas um número constante de segmentos da apresentação.

Tal afirmação só pode ser provada para apresentações sobre domínios suficientemente grandes em relação ao grau do polinômio. Arora e Safra [AS92] provaram para domínios de tamanho cúbico em *de*. Sudan [Sud92] melhorou isso para quadrático. Polishchuk e Spielman [PS94] conseguiram provar que isso é possível para domínios lineares, isto é, cujo tamanho é maior que *de* por apenas um fator constante. Isso foi essencial para obter as provas holográficas de tamanho quase-linear.

Neste trabalho não será mostrada toda a parte algébrica para a obtenção desses resultados. Vimos acima referências para os locais onde se podem encontrar essas demonstrações. Em particular, o último resultado também é explicado em [Spi95], onde é descrito com mais detalhes do que no artigo original [PS94]. Aqui apenas são enunciados os teoremas principais, a serem utilizados nos Capítulos 7 e 8.

Por motivo de clareza, vamos definir o seguinte termo, que é usado em várias partes deste trabalho.

Definição 4.7 Sortear $a \in C$ *significa escolher uniformemente ao acaso um elemento a de um conjunto C.*

Quando o sorteio é de vários elementos de um conjunto, cada escolha é feita independentemente.

O algoritmo a seguir checa se uma (d, e)-apresentação sobre um domínio $X \times Y$ é uma apresentação correta de algum polinômio de grau (d, e).

Algoritmo 4.8 Checagem para Duas Variáveis

1. *Sortear um ponto* $(x_0, y_0) \in X \times Y$;

2. *Ler o valor* v *atribuído ao ponto* (x_0, y_0);
   Ler a string p_y *atribuída a* x_0;
   Ler a string p_x *atribuída a* y_0;

3. *Aceitar se* p_x *representa um polinômio tal que* $p_x(y_0) = v$
   e p_y *representa um polinômio tal que* $p_y(x_0) = v$.

Observemos que, embora não se teste explicitamente que os polinômios p_x e p_y possuam o grau correto, esse teste, na verdade, está subentendido dentro do algoritmo. Isto ocorre porque os dados são uma seqüência de bits (reveja Def. 4.5) e tanto o tamanho quanto a localização (endereçamento) dos segmentos são tratados pelo algoritmo *como se* os polinômios tivessem os graus desejados.

Se a apresentação for correta, então, por definição, existe um polinômio tal que, no algoritmo acima, v é o valor desse polinômio em (x_0, y_0), p_x é a sua restrição aos pontos (x, y_0) e p_y é a restrição a (x_0, y). Portanto, é fácil perceber que o algoritmo sempre aceita as entradas que são apresentações corretas.

A recíproca também é verdadeira, como se pode ver na seguinte proposição, que é bastante conhecida.

Proposição 4.9 *Sejam* $X = \{x_1, \ldots, x_m\}$, $Y = \{y_1, \ldots, y_n\}$, *e sejam* $d < m$, $e < n$. *Seja* $f(x, y)$ *uma função em* $X \times Y$ *tal que, para* $1 \leq j \leq n$, $f(x, y_j)$ *coincide em* X *com algum polinômio de grau* d *em* x, *e, para* $1 \leq i \leq m$, $f(x_i, y)$ *coincide em* Y *com algum polinômio de grau* e *em* y. *Então, existe um polinômio* $P(x, y)$ *de grau* (d, e) *tal que* $f(x, y)$ *coincide com* $P(x, y)$ *em todos os lugares de* $X \times Y$.

Prova. Basta construir P usando interpolação de Lagrange. □

Essa proposição significa o seguinte. Se esse algoritmo aceita uma entrada com probabilidade 1, então ela de fato corresponde à apresentação correta de algum polinômio de grau (d, e).

Porém, é claro que, sem ler *todos* os dados, é impossível saber se a probabilidade de aceitar vale 1. Por isso, gostaríamos de um resultado que afirmasse algo do seguinte tipo: se o algoritmo aceita uma (d, e)-apresentação A com probabilidade próxima a 1, então A deve ser muito parecida com a apresentação de algum polinômio de grau (d, e).

Utilizaremos o seguinte conceito de proximidade.

Definição 4.10 *Duas apresentações de mesmo grau sobre o mesmo domínio são* ε-próximas *se diferem, em cada uma das três listas que compõem as apresentações, por no máximo uma fração* ε *dos seus segmentos.*

Definição 4.11 *Uma* (d, e)*-apresentação* A *sobre um domínio* $X \times Y$ *é* ε-boa *se* A *é* ε*-próxima à apresentação correta de algum polinômio* p*, de grau* (d, e)*, sobre* $X \times Y$*.*

Notemos que, se A é uma apresentação ε-boa para um ε suficientemente pequeno, então existe um único polinômio p cuja apresentação é próxima de A.

Proposição 4.12 *Seja* A *uma apresentação* ε*-boa de grau* (d, e) *sobre um domínio* $X \times Y$*. Se*

$$\varepsilon < \frac{1}{2}\left\{1 - \left(\frac{d}{|X|} + \frac{e}{|Y|}\right)\right\}$$

então existe um único polinômio p *tal que* A *e a apresentação correta de* p *diferem no máximo em uma fração* ε *dos seus segmentos.*

Prova. Vamos assumir, como via de contradição, que existam dois polinômios p e p' distintos cujas apresentações corretas coincidem com A em todos exceto em uma fração ε dos seus segmentos. Então p e p' possuem o mesmo valor em pelo menos uma fração $(1 - 2\varepsilon)$ do domínio. Logo, o polinômio $p - p'$ é zero em pelo menos $|X||Y|(1 - 2\varepsilon)$ pontos.

Utilizando a hipótese sobre ε vemos que o número de raízes em $p - p'$ é pelo menos

$$\begin{aligned} |X||Y|(1-2\varepsilon) &> |X||Y|\left(1 - 2 \cdot \frac{1}{2}\left\{1 - \left(\frac{d}{|X|} + \frac{e}{|Y|}\right)\right\}\right) \\ &\geq |X||Y|\left(\frac{d}{|X|} + \frac{e}{|Y|}\right) \end{aligned}$$

e, portanto, pelo Lema 4.6, o polinômio $p - p'$ seria o polinômio nulo, contradizendo a hipótese de que p e p' sejam distintos. □

Antes de enunciar o próximo resultado, vamos rever a definição de apresentação correta (Def. 4.4), com $|X| = m$ e $|Y| = n$, e observá-la de outro ponto de vista. Chamemos de L_1, L_2 e L_3, respectivamente, as três listas que aparecem nessa definição.

A lista L_2 é composta de polinômios de uma variável de grau e em y. Podemos interpretá-la como uma função de X em $\mathcal{G}^{e+1}$ pois, para cada $x_0 \in X$, L_2 associa um polinômio, que é representado por $e + 1$ coeficientes.

Outra maneira de olhar para L_2 é como uma função em x e y que é arbitrária na direção x mas que, na direção y, é sempre parecida com um polinômio de grau e.

Sabemos que qualquer função no domínio X pode ser representada como um polinômio de grau $|X| - 1 = m - 1 \leq m$. Portanto L_2 pode ser vista como a descrição de um polinômio $C(x, y)$ de grau $(m. e)$. Analogamente, L_3 descreve um polinômio $R(x, y)$ de grau (d, n).

Com esse ponto de vista. podemos enunciar um importante teorema. Sua demonstração (que se encontra nas referências já citadas) é longa e engenhosa.

Teorema 4.13 *Seja $\mathcal{G}$ um corpo e sejam dados conjuntos $X = \{x_1, \ldots, x_m\} \subseteq \mathcal{G}$ e $Y = \{y_1, \ldots, y_n\} \subseteq \mathcal{G}$. Sejam $R(x, y)$ um polinômio sobre $\mathcal{G}$ de grau (d, n) e $C(x, y)$ um polinômio sobre $\mathcal{G}$ de grau $(m. e)$. Se*

$$2\left(\frac{d}{m} + \frac{e}{n} + \delta\right) < 1$$

e

$$\Pr_{(x,y)\in X\times Y}[R(x, y) \neq C(x, y)] < \delta^2,$$

então existe um polinômio $Q(x, y)$ de grau (d, e) tal que

$$\Pr_{(x,y)\in X\times Y}[R(x, y) \neq Q(x, y) \quad \textit{ou} \quad C(x, y) \neq Q(x, y)] < 2\delta^2.$$

Para ver a utilidade desse teorema, voltemos ao Algoritmo de Checagem para Duas Variáveis, cuja entrada é uma apresentação A, com polinômios R e C conforme a discussão anterior.

Se A for aceita com alta probabilidade pelo algoritmo, então $R(x, y)$ e $C(x, y)$ coincidem na maior parte do domínio $X \times Y$. Isso satisfaz a hipótese do teorema acima, e concluímos que A é próxima à apresentação correta de algum polinômio $Q(x, y)$ de grau (d, e).

Isso está formalizado nos seguintes enunciados.

Lema 4.14 *Seja A uma (d, e)-apresentação sobre um domínio $X \times Y$ tal que $|X| > 8d$ e $|Y| > 8e$. Se a probabilidade de o Algoritmo de Checagem para Duas Variáveis aceitar é maior que $1 - \varepsilon$, para $\varepsilon < 1/16$, então a apresentação é 3ε-boa.*

Prova. Segue do Teorema 4.13, com $\delta = 1/4$. □

Corolário 4.15 *O código das (d, e)-apresentações sobre o domínio $X \times Y$ possui um checador que lê apenas um número constante de segmentos da sua entrada.*

Prova. O checador é obtido rodando-se um número constante de vezes o Algoritmo de Checagem para Duas Variáveis. Essa constante é escolhida de acordo com a probabilidade máxima de erro desejada.

Quanto aos segmentos, basta constatar (veja texto após Def. 4.4) que cada string p_x ou p_y, lida pelo algoritmo, está em um único segmento. □

4.3 Polinômios de três variáveis

Vamos construir codificações de polinômios de três variáveis. Elas serão o principal componente da nossa prova holográfica, que é apresentada nos Capítulos 7 e 8. Para trabalhar com esses polinômios, precisamos dos teoremas que estão na seção anterior, relativos a polinômios de duas variáveis.

Começamos com os conceitos relativos às apresentações. Nada mais são que a extensão natural para três variáveis dos conceitos vistos na seção anterior.

Seja $p(x, y, z)$ um polinômio de grau (d_x, d_y, d_z) sobre um corpo $\mathcal{G}$. Usaremos as restrições de p a cada uma das linhas paralelas aos eixos passando pelo domínio $X \times Y \times Z$.

Notação 4.16 *A restrição do polinômio $p(x, y, z)$ aos pontos $x = x_0$ e $y = y_0$ nos dá um polinômio de grau d_z, em uma variável. Esse polinômio será denotado por $p|_{(x_0,y_0,\cdot)}$. Usamos notações análogas, $p|_{(x_0,\cdot,z_0)}$ e $p|_{(\cdot,y_0,z_0)}$, para os polinômios com restrições nas outras variáveis.*

Definição 4.17 *Sejam dados X, Y, $Z \subseteq \mathcal{G}$. Uma* apresentação correta *de p sobre $X \times Y \times Z$ consiste em*

(i) uma lista com os valores de p em cada ponto de $X \times Y \times Z$;

(ii) para cada $(x_0, y_0) \in X \times Y$, os coeficientes de $p|_{(x_0,y_0,\cdot)}$;

(iii) para cada $(x_0, z_0) \in X \times Z$, os coeficientes de $p|_{(x_0,\cdot,z_0)}$;

(iv) para cada $(y_0, z_0) \in Y \times Z$, os coeficientes de $p|_{(\cdot,y_0,z_0)}$.

Definimos também uma (d_x, d_y, d_z)-*apresentação* de forma análoga à Def. 4.5, ou seja, possui o mesmo tamanho mas pode não ser uma apresentação correta. Similarmente ao Lema 4.6, prova-se o seguinte resultado, para apresentações de três variáveis.

Lema 4.18 *Seja p um polinômio de grau (d_1, d_2, d_3) sobre $\mathcal{D} = \mathcal{D}_1 \times \mathcal{D}_2 \times \mathcal{D}_3$. Se p tem mais do que*

$$|\mathcal{D}| \left(\frac{d_1}{|\mathcal{D}_1|} + \frac{d_2}{|\mathcal{D}_2|} + \frac{d_3}{|\mathcal{D}_3|} \right)$$

raízes em $\mathcal{D}$, então p é o polinômio nulo.

Uma conseqüência é que, sobre domínios suficientemente grandes, as apresentações de polinômios p' e p'' distintos estão bastante distanciadas entre si. Isso segue do lema anterior, pela consideração das raízes do polinômio diferença entre p' e p''.

O teorema para a checagem de polinômios de três variáveis é um corolário do Teorema 4.13.

Teorema 4.19 *Sejam $P_1(x, y, z)$, $P_2(x, y, z)$ e $P_3(x, y, z)$ polinômios sobre um corpo $\mathcal{G}$ cujos graus são $(d, 12e, 12c)$, $(12d, e, 12c)$ e $(12d, 12e, c)$ respectivamente. Sejam X, Y, Z subconjuntos de $\mathcal{G}$ de tamanhos $12d$, $12e$, $12c$ respectivamente. Se*

$$\Pr_{(x,y,z) \in X \times Y \times Z} [P_1(x, y, z) = P_2(x, y, z) = P_3(x, y, z)] > 1 - \frac{\delta^2}{2916},$$

para $\delta^2 < 1$, então existe um polinômio $Q(x, y, z)$ de grau (d, e, c) tal que

$$\Pr_{(x,y,z) \in X \times Y \times Z} [Q(x, y, z) = P_1(x, y, z) = P_2(x, y, z) = P_3(x, y, z)] > 1 - \frac{2\delta}{9}.$$

Prova. [Sucinta] Para ir do teorema de duas variáveis para este com três variáveis, primeiro aplicamos o Teorema 4.13 em cada plano perpendicular ao eixo y, e obtemos uma coleção $R^{y_i}(x, z)$ de polinômios de duas variáveis em (x, z) de grau (d, c). Se tratarmos cada elemento dessa coleção como um polinômio de uma variável em x que assume valores que são funções em z, veremos $R^{y_i}(x)$ como uma coleção de polinômios sobre o corpo $\mathcal{G}(z)$. Finalmente, vista como um todo, essa coleção é uma função $R(x, y)$ que, na direção y é uma função arbitrária mas, na direção x sempre se comporta como um polinômio de grau d.

De modo análogo, a aplicação do teorema de duas variáveis aos planos perpendiculares ao eixo x nos dá uma função $C(x, y)$ que, na direção y, comporta-se como polinômios de grau e.

Esses polinômios $R(x, y)$ e $C(x, y)$ sobre $\mathcal{G}(z)$ satisfazem as hipóteses para mais uma aplicação do Teorema 4.13, de onde obtemos o polinômio Q como desejado. □

Esse teorema nos permite afirmar que há uma versão do Lema 4.14 que se aplica a apresentações de três variáveis. o que nos leva ao seguinte resultado.

Corolário 4.20 **[Algoritmo de Checagem para Três Variáveis]** *Existe um algoritmo para checagem de apresentações de polinômios com três variáveis, que é análogo ao Algoritmo 4.8, e que lê apenas um número constante de segmentos da apresentação.*

4.4 Sub-apresentações e verificação

É claro que uma apresentação correta de um polinômio em um domínio $\mathcal{D}$ contém apresentações corretas desse mesmo polinômio em subconjuntos de $\mathcal{D}$. Nesta seção vemos como lidar com essas sub-apresentações.

Começamos com uma observação simples.

Proposição 4.21 *Seja A uma (d_x, d_y, d_z)-apresentação ε-boa sobre um domínio $\mathcal{D} = X \times Y \times Z$ e seja $\mathcal{D}' = X' \times Y' \times Z'$ um subconjunto de $\mathcal{D}$ tal que $c|\mathcal{D}'| \geq |\mathcal{D}|$ para uma constante $c \geq 1$. Se tomamos A' composta pelos elementos de A que correspondem a uma apresentação sobre $\mathcal{D}'$, então A é $c\varepsilon$-boa.*

Além disso, se

$$\frac{d_x}{|X|} + \frac{d_y}{|Y|} + \frac{d_z}{|Z|} + 4c\varepsilon < 1,$$

então A e A' são próximas à apresentação correta do mesmo polinômio de grau (d_x, d_y, d_z).

Prova. A primeira parte é óbvia. A segunda segue do Lema 4.18. □

Um outro tipo de sub-apresentação é o de dimensão mais baixa. Por exemplo, uma apresentação de três variáveis A sobre $X \times Y \times Z$ contém uma apresentação de duas variáveis sobre $X \times Y \times z_0$, para $z_0 \in Z$. Mas o fato de que A seja ε-boa não implica muita coisa sobre essa sub-apresentação.

Para mostrar que a sub-apresentação pode ser certificada como boa, mostraremos uma propriedade mais forte das apresentações: elas possuem verificadores simples. Isto é, se sabemos que A é ε-boa, para um ε suficientemente pequeno, podemos checar se qualquer particular segmento de A é o mesmo que o segmento da apresentação correta à qual A é próxima.

Algoritmo 4.22 **Verificação para Três Variáveis**
Tomemos ε alguma constante pequena.

Entrada do algoritmo: uma apresentação A sobre $X \times Y \times Z$;
um ponto (x_0, y_0, z_0).

1. *Checar se a apresentação é ε-boa.*

2. *Checar se a sub-apresentação sobre $X \times Y \times z_0$ é ε-boa.*

3. *Sortear um número constante de pontos de $X \times Y \times z_0$ e checar se o polinômio de uma variável de A em z que passa por cada ponto assume o valor atribuído a esse ponto pela apresentação.*

4. *Sortear um número constante de pontos de $X \times y_0 \times z_0$ e checar se o polinômio de uma variável de A em y que passa por cada ponto assume o valor atribuído a esse ponto pela apresentação.*

5. *Checar se o polinômio de uma variável de A em x que passa por (x_0, y_0, z_0) coincide com o valor atribuído a esse ponto por A.*

Um algoritmo de verificação similar a este apareceu em [Sud92], e rodava em domínios de tamanho quadrático. A demonstração do Teorema 4.19 em [PS94] implica que agora podemos trabalhar sobre domínios cujo tamanho é linear no grau das apresentações.

Lema 4.23 *Existe uma constante c tal que, para todo $\varepsilon > 0$, existem constantes no Algoritmo de Verificação para Três Variáveis tais que se A é uma (d_x, d_y, d_z)-apresentação sobre um domínio $X \times Y \times Z$ onde $cd_x < |X|$, $cd_y < |Y|$ e $cd_z < |Z|$, então*

- *Se o Passo 1 do algoritmo de verificação aceita com probabilidade maior que $1/2$, então existe um polinômio $p(x, y, z)$ de grau (d_x, d_y, d_z) tal que A é ε-próxima de uma apresentação correta de $p(x, y, z)$.*

- *Se os Passos 1, 2 e 3 aceitam com probabilidade maior que $1/2$, então a sub-apresentação de P em $X \times Y \times z_0$ é ε-próxima de uma apresentação correta de $p(x, y, z_0)$.*

- *Se os Passos 1 a 4 aceitam com probabilidade maior que $1/2$, então o polinômio de uma variável de A em x que passa pela linha $(\cdot, y_0, z_0)$ é de fato $p(x, y_0, z_0)$.*

- *Se os Passos 1 a 5 aceitam com probabilidade maior que $1/2$, então o valor que A atribui ao ponto (x_0, y_0, z_0) é $p(x_0, y_0, z_0)$.*

Prova. Esses fatos seguem do Lema 4.18, dos Corolários 4.15 e 4.20, e do Teorema 4.19. □

Com esse lema, portanto, podemos afirmar que as codificações de polinômios de três variáveis são verificáveis pela consulta a apenas um número constante de segmentos. O

fato de ser verificável (conforme a Def. 4.2) significa que, além de certificar que A está próxima de uma apresentação correta, o checador pode conferir posições individuais (!) dentro de A e ter certeza de que elas contêm valores iguais aos da apresentação correta. Note que é surprendente obter tal propriedade lendo tão poucas informações contidas na apresentação.

Capítulo 5

Um Problema $\mathcal{NP}$-completo

Precisamos de um problema $\mathcal{NP}$-completo para servir de fundamento ao nosso sistema de provas holográficas. Neste capítulo, veremos a descrição de um grafo relacionado com o Grafo de De Bruijn, e mostraremos como o problema $\mathcal{NP}$-completo de "Satisfazibilidade de Circuitos" pode ser reduzido a um problema de coloração nesse grafo. Uma característica importante é que é particularmente fácil criar um sistema de provas holográficas para esse problema de coloração.

Para começar, vejamos o motivo porque não criamos um sistema de provas holográficas diretamente para o problema de Satisfazibilidade de Circuitos (CircSat, ver Def. 2.4). A maneira habitual de descrever uma instância do CircSat é fornecer, para cada porta no circuito, a sua função e os nomes das suas entradas. Não é simples checar se uma atribuição satisfaz um circuito, pois, para calcular o valor de uma porta, é necessário primeiro ler os nomes das entradas dessa porta para, só então, ler os seus valores. Tais portas podem estar em qualquer lugar do circuito e, em conseqüência, o projeto de um sistema de provas holográficas diretamente para o CircSat faz com que a maior parte da prova seja dedicada a demonstrar que os valores nas portas são corretamente movidos entre as posições da prova. Devido a esse *overhead*, evitamos buscar um sistema eficiente de provas holográficas para esse problema.

Vamos confeccionar um outro problema tal que, quando quisermos determinar se uma de suas restrições é satisfeita, saberemos imediatamente onde devem ser buscados os valores que precisamos examinar. Nesse problema, computamos os nomes das variáveis envolvidas em uma restrição simplesmente a partir do nome dessa restrição; e é dessa forma que nos livramos do *overhead* anteriormente citado.

Também será fácil reduzir instâncias do CircSat a instâncias do nosso problema. Além disso, outra característica importante é a que se vê no Cap. 6, onde esse problema também consegue ser convertido em um objeto bastante simples do ponto de vista

algébrico.

O problema que usaremos será a coloração de um certo grafo descrito na Seção 5.1. Vemos na Seção 5.2 que uma instância desse problema consiste em uma primeira atribuição de cores a cada nó do grafo. Na Seção 5.3 vemos que uma solução do problema será uma segunda atribuição de cores aos nós de modo que os pares de cores atribuídas a cada nó e a seus vizinhos satisfazem certas regras de coloração.

5.1 Grafos de De Bruijn

Vamos mostrar o que é um grafo de De Bruijn circular. Sobre tal tipo de grafo é que, adiante, descrevemos nosso problema de coloração.

Neste texto estamos trabalhando principalmente com circuitos booleanos. Para evitar confusão de nomes chamaremos *grafo-circuito* ao grafo que é um circuito (caminho fechado sem repetição de nós).

Definição 5.1 *O* grafo de De Bruijn B_n *é um grafo orientado com* 2^n *nós, no qual cada nó é representado por uma string binária com* n *dígitos. O nó representado pela string* $(x_1, \ldots, x_n)$ *tem arestas apontando para os nós representados por*

$$(x_2, \ldots, x_n, 0) \quad e \quad (x_2, \ldots, x_n, 1).$$

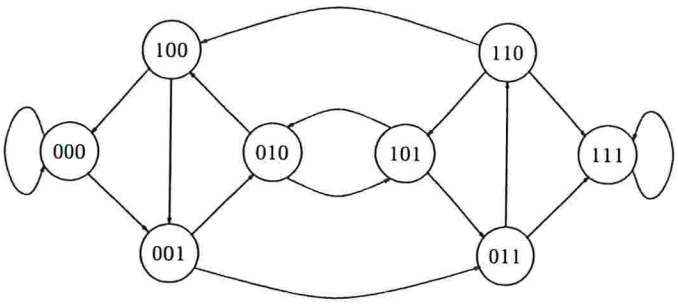

Figura 5.1: Grafo de De Bruijn B_3

A seguir definimos grafo de De Bruijn circular, que é o produto de um grafo de De Bruijn com um grafo-circuito; nesse produto, existe uma aresta do vértice (x, a) para o vértice (y, b) se e somente se houver uma aresta de x para y e de a para b. Veja Fig. 5.2. O tamanho do grafo-circuito precisa ser alguma constante vezes n que seja grande o bastante para nós realizarmos certas operações de roteamento no grafo. A escolha de $5n$ será suficiente.

Definição 5.2 *O* grafo de De Bruijn circular D_n *é um grafo orientado com* $5n \cdot 2^n$ *nós no qual cada nó é representado por um par consistindo em uma string binária com* n *dígitos e um número módulo* $5n$. *O nó representado pelo par* $((x_1, \ldots, x_n), a)$ *tem arestas apontando para os nós representados por*

$$((x_2, \ldots, x_n, 0), a+1) \quad e \quad ((x_2, \ldots, x_n, 1), a+1),$$

onde a soma $a+1$ *é tomada módulo* $5n$.

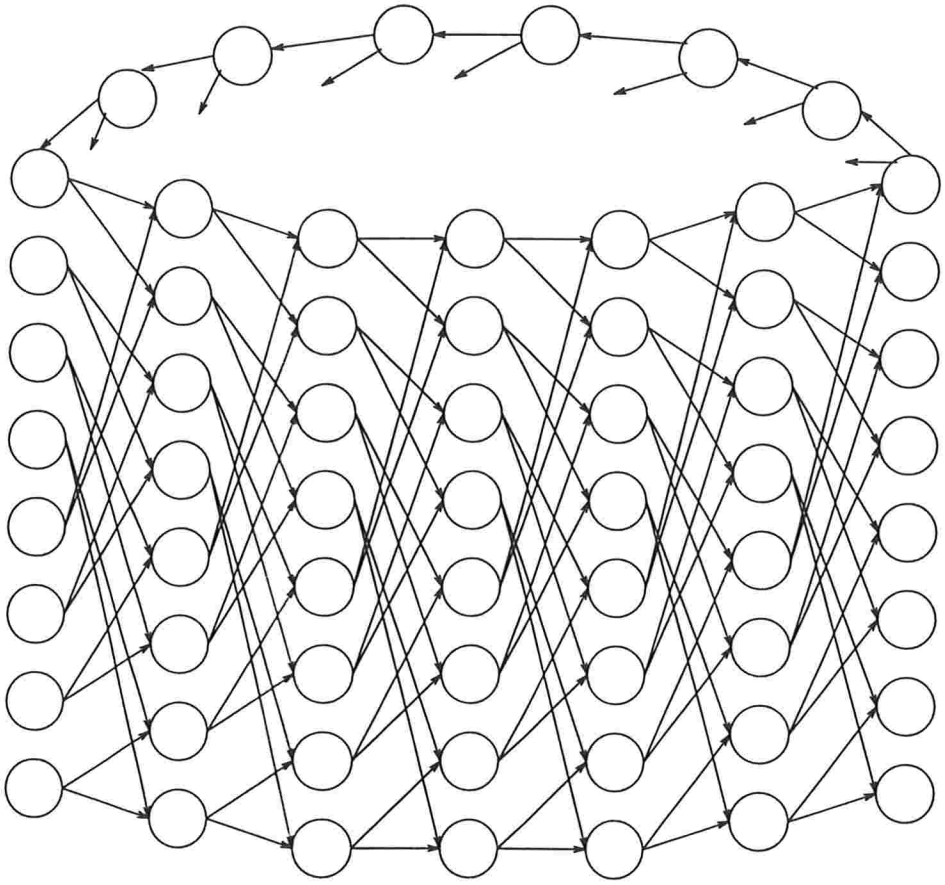

Figura 5.2: Grafo de De Bruijn circular

Em D_n, para cada $a \in \{0, \ldots, 5n-1\}$, chamaremos uma *coluna* do grafo ao conjunto dos nós cuja representação termina com a.

Há dois motivos porque usamos grafos de De Bruijn circulares. Um é o fato já comentado de que possuem uma descrição algébrica muito simples. O outro é que pode-se rotear qualquer permutação em um grafo desse tipo; e isso nos permite "desenhar" um circuito booleano *em* um grafo, conforme veremos na próxima seção.

5.2 Instância do problema

Vamos descrever um modo de "desenhar" um circuito booleano dentro de um grafo de De Bruijn circular, e vamos fabricar um problema de coloração tal que o grafo possui uma coloração válida se e somente se existir alguma entrada (atribuição de 0's e 1's) que satisfaz ao circuito dado.

Um jeito de representar um circuito booleano é fazer uma figura contendo as entradas, as portas e as linhas significando os "fios" que as conectam. Por exemplo, na Fig. 5.3 está o circuito correspondente à fórmula booleana $(x_1 \wedge x_3) \vee (\neg x_3 \wedge x_2)$. Para facilitar futuros argumentos, acrescentamos uma porta extra para a saída do circuito.

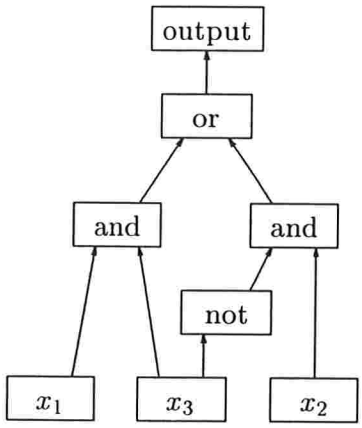

Figura 5.3: Um circuito booleano

Consideremos agora um circuito C com m_1 portas binárias (inclusive a porta *output*) e m_2 entradas. Seja r o menor inteiro tal que $2^r \geq m_1 + m_2$. Vamos desenhar o circuito C dentro do grafo de De Bruijn circular D_r, isto é, um grafo com $5r2^r$ nós, sendo $5r$ o número de colunas.

Para isso vamos escolher uma coluna do grafo D_r e associar a cada nó dessa coluna um nó do circuito C. Isso se faz atribuindo a cada nó uma das cores do conjunto {AND, OR, NOT, INPUT, OUTPUT, λ} para indicar o tipo de porta a que corresponde esse nó do grafo. A cor λ indica "vazio" e significa que o nó sobrou sem ser associado a nenhuma porta do circuito. As entradas do circuito serão também chamadas de portas (no caso, portas de cor INPUT).

Nas outras colunas do grafo os nós serão coloridos de uma maneira que descreva as conexões que existem entre as portas do circuito. Para cada um desses nós associamos uma *ação de chaveamento* do tipo das que estão representadas na Fig. 5.4. Essa associação se faz atribuindo ao nó uma cor; cada cor representa um entre nove possíveis

tipos de chaveamento. Notemos que um dos chaveamentos permitidos é tomar uma das mensagens que chegam e enviá-la para ambas as saídas (por exemplo, a chave (c) na Fig. 5.4).

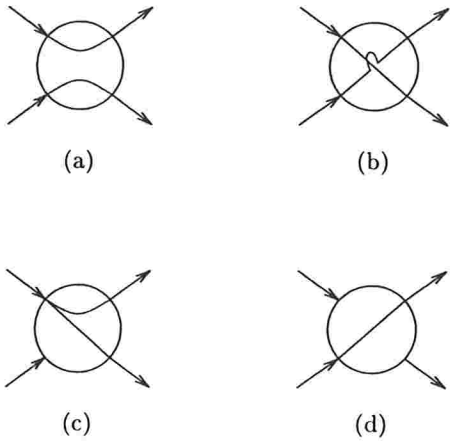

Figura 5.4: Algumas ações de chaveamento

Queremos atribuir ações de chaveamento de modo a conectar cada porta às suas entradas. Isso pode ser visto como um *problema de roteamento* em que há no máximo dois pacotes chegando a cada nó (pois as portas AND e OR são binárias). Por outro lado, a saída de uma porta do circuito pode servir de entrada para várias outras; isso significa que, no roteamento, um mesmo pacote deve poder ser levado para vários lugares.

Pela utilização de técnicas que são padrão em roteamento de pacotes (veja [Lei92]) sabemos que $5r$ passos através de um grafo de De Bruijn de 2^r nós são suficientes para resolver esse problema.

No nosso caso, a coluna associada às portas contém ao mesmo tempo os nós de origem e os de destino. O resultado de [Lei92] significa que é possível encontrar ações de chaveamento para cada nó das outras colunas de D_r que estabeleçam no grafo caminhos sem colisão que permitam que a saída de cada porta consiga ser transportada ao longo das outras $5r-1$ colunas do grafo circular e chegue às entradas das portas que precisam desse valor, na coluna destino. Veja Fig. 5.5: as colunas da esquerda e da direita do grafo devem ser identificadas uma com a outra, mas estão representadas duas vezes por conveniência da figura. As linhas escuras correspondem a fios no circuito.

Recapitulando: em primeiro lugar recebemos uma instância do CircSat, isto é, um circuito booleano C; então atribuímos cores (de 6 tipos) aos nós de uma coluna do grafo para associá-los às portas de C; e atribuímos cores (de 9 tipos) aos nós das colunas restantes para associá-los a ações de chaveamento. O número total de cores necessárias é pequeno e não depende do tamanho de C. Essa primeira coloração do grafo será

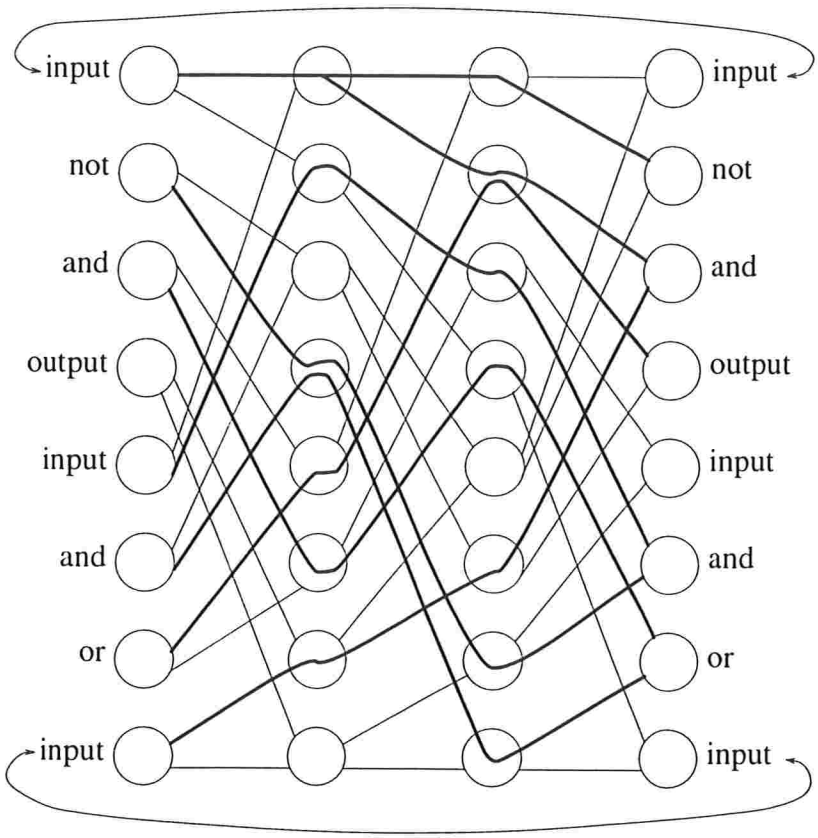

Figura 5.5: O desenho de um circuito dentro de um grafo de De Bruijn circular

chamada *instância do problema de coloração.*

É importante observar que o tamanho desse tipo de descrição de um circuito não é muito diferente do tamanho de uma descrição mais tradicional. Para um circuito onde o número de portas é $m_1 + m_2 = m$, a descrição usual costuma dar um nome (tal nome pode ser simplesmente um endereço) para cada porta. Estes nomes gastam $\log m$ bits. Além disso, para cada porta é preciso listar a operação que ela realiza e os nomes das portas que são suas entradas. Portanto a descrição do circuito possui tamanho total $\Theta(m \log m)$. No nosso caso é desnecessário atribuir nomes às portas pois suas conexões são dadas pela rede de roteamento. Como nossa descrição se faz atribuindo a cada nó do grafo um entre um número constante de cores, então o tamanho dessa descrição é da ordem do tamanho do grafo, $O(2^r 5r)$, ou seja, $O(m \log m)$; portanto, difere do tamanho da descrição convencional apenas por um fator constante.

5.3 Solução do problema

Uma prova de que um circuito é satisfazível consiste em uma atribuição de 0's e 1's às entradas e portas do circuito, que seja consistente com uma atribuição satisfatória. Veja Fig. 5.6. A representação dessa prova como uma segunda coloração do grafo consistirá em uma atribuição de 0's e 1's para as arestas entrando e saindo dos nós do grafo, que seja consistente com a atribuição, com as ações das portas e com as ações de chaveamento. Entretanto, não vamos colorir as arestas do grafo mas sim os nós. Por isso a prova, na realidade, vai atribuir uma quádrupla de símbolos a cada nó, dizendo se as arestas que chegam e que saem desse nó são 0, 1 ou $\not{b}$ (em branco). A Fig. 5.8 contém exemplos válidos de segundas cores. O número de cores (ou seja, de quádruplas) diferentes é 3^4; portanto, tal como a instância, a prova também pode ser descrita usando um número finito de cores. Essa segunda coloração, representando uma prova, é chamada *solução do problema de coloração.*

Vamos escolher regras de coloração de modo que as únicas colorações válidas para o grafo sejam as que seguem as seguintes condições:

(i) atribuir 0 ou 1 a cada nó de tipo INPUT;

(ii) calcular corretamente o valor das saídas dos nós do tipo OR, AND, NOT;

(iii) propagar os valores ao longo das arestas seguindo corretamente as ações de chaveamento;

(iv) produzir 1 no nó OUTPUT.

A Fig. 5.7 contém um desenho de uma prova de que o circuito da Fig. 5.3 é satisfazível. A Fig. 5.9 contém representações de pedaços de segundas colorações que violam as regras de coloração.

Para resumir, nosso problema de coloração é, dada uma primeira atribuição de cores ao grafo, encontrar uma segunda atribuição que satisfaça a todas as regras de coloração.

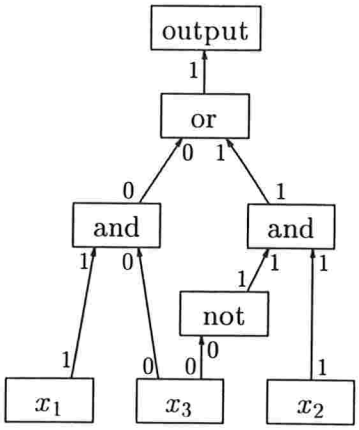

Figura 5.6: Uma prova de que um circuito é satisfazível

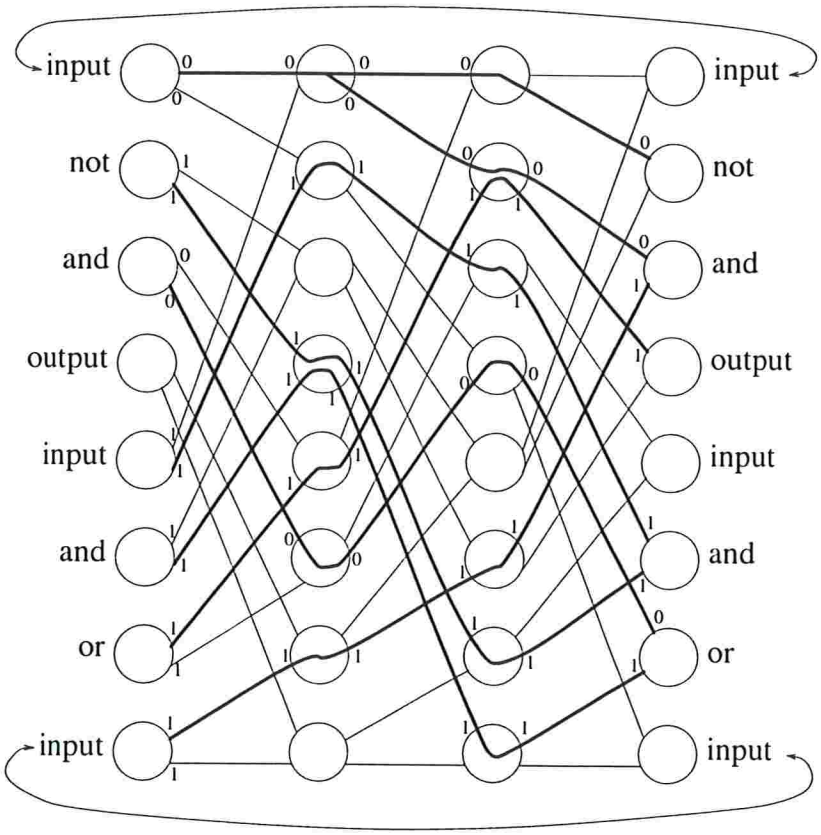

Figura 5.7: Outra prova de que um circuito é satisfazível

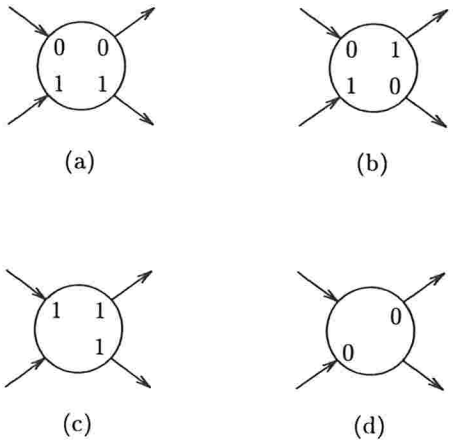

Figura 5.8: Algumas "segundas cores" válidas para os chaveamentos da Figura 5.4

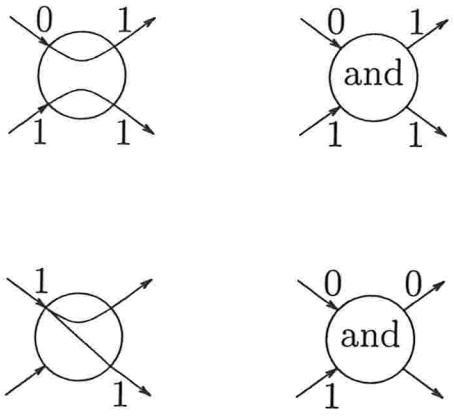

Figura 5.9: Atribuições de "segundas cores" que violam as regras de coloração

Capítulo 6

Aritmetização

A característica essencial das provas holográficas é que podem ser checadas probabilisticamente através do exame de uma quantidade pequena de suas posições. Para tanto, o principal ingrediente é o uso de codificações baseadas em polinômios. A aritmetização que veremos aqui tem como objetivo expressar, através de funções polinomiais, o problema $\mathcal{NP}$-completo descrito no Capítulo 5.

6.1 Descrição algébrica do grafo

Vamos mostrar que os grafos de De Bruijn circulares possuem uma descrição algébrica muito simples, que nos permite identificar os nós do grafo com pontos em um corpo de tal modo que os nomes dos vizinhos de um nó podem ser expressos como um polinômio de grau baixo nos nomes desse nó.

Na verdade, a descrição algébrica que obtemos é para o grafo que chamamos *grafo de De Bruijn estendido* que é o produto do grafo-circuito de $5n+1$ nós com o grafo de De Bruijn. A identificação entre a primeira e a última colunas é fornecida pela prova holográfica. Vamos usar essa descrição para traduzir nosso problema de coloração de grafo para um problema algébrico no qual podemos aplicar os instrumentos desenvolvidos no Cap. 4. Começamos definindo um grafo nos elementos de $\mathrm{GF}(2^n)$.

Definição 6.1 *Um* grafo de Galois G_n *é um grafo orientado com* 2^n *nós no qual cada nó é identificado com um elemento de* $GF(2^n)$. *Seja* α *um gerador de* $GF(2^n)$. *O nó representado por* $\gamma \in GF(2^n)$ *possui arestas apontando para os nós representados por*

$$\alpha\gamma \quad e \quad \alpha\gamma + 1.$$

Lema 6.2 *O grafo de Galois* G_n *é isomorfo ao grafo de De Bruijn* B_n.

Prova. Começamos recordando uma representação padrão de $\mathrm{GF}(2^n)$ (no sentido não-técnico de "representação"). É fácil de ver que $\mathrm{GF}(2^n)$ é um espaço vetorial de dimensão n sobre $\mathrm{GF}(2)$. Ademais, o fato de α gerar o grupo multiplicativo de $\mathrm{GF}(2^n)$ implica que

$$1, \alpha, \ldots, \alpha^{n-1}$$

geram $\mathrm{GF}(2^n)$ sobre $\mathrm{GF}(2)$. Daí segue que existe um polinômio $p(X) \in \mathrm{GF}(2)[X]$ de grau n tal que

$$\mathrm{GF}(2^n) = \mathrm{GF}(2)[X]/p(X).$$

Ou seja, os elementos de $\mathrm{GF}(2^n)$ podem ser representados como polinômios em α de grau no máximo $n-1$ com coeficientes em $\mathrm{GF}(2)$ (aqui e a seguir identificamos α e X). A adição desses polinômios é realizada componente a componente. Para entender o que acontece na multiplicação, tomemos

$$p(\alpha) = \alpha^n + c_1\alpha^{n-1} + \cdots + c_{n-1}\alpha + c_n.$$

Multiplicamos elementos de $\mathrm{GF}(2^n)$ pelo algoritmo normal de multiplicação de polinômios, lembrando de fazer a soma componente a componente. Se obtivermos um polinômio de grau maior que $n-1$, aplicamos a relação

$$\begin{aligned} \alpha^n &= -(c_1\alpha^{n-1} + \cdots + c_{n-1}\alpha + c_n) \\ &= c_1\alpha^{n-1} + \cdots + c_{n-1}\alpha + c_n, \end{aligned}$$

até obter um polinômio de grau menor ou igual a $n-1$.

Vamos usar vetores binários de tamanho n para representar cada um dos 2^n elementos do grafo de De Bruijn. Definimos ϕ uma função dos vértices de B_n nos vértices de G_n, da seguinte forma

$$\phi(b_1, b_2, \ldots, b_n) = \alpha^{n-1}b_1 + \alpha^{n-2}(b_2 + c_1b_1) + \cdots + \left(b_n + \sum_{i=1}^{n-1} c_i b_{n-i}\right).$$

É fácil verificar que ϕ é uma bijeção. Para ver que é um isomorfismo de grafos, analisemos se ϕ preserva arestas. As arestas que saem de $(b_1, \ldots, b_n)$ vão para os vértices

$$(b_2, b_3, \ldots, b_n, 0) \quad \text{e} \quad (b_2, b_3, \ldots, b_n, 1),$$

no grafo B_n. Por outro lado, em G_n, uma aresta que sai de

$$\alpha^{n-1}b_1 + \alpha^{n-2}(b_2 + c_1b_1) + \cdots + \left(b_n + \sum_{i=1}^{n-1} c_i b_{n-i}\right)$$

vai para

$$\begin{aligned} & \alpha\left(\alpha^{n-1}b_1+\alpha^{n-2}(b_2+c_1b_1)+\cdots+\left(b_n+\sum_{i=1}^{n-1}c_ib_{n-i}\right)\right) \\ = \; & b_1\left(c_1\alpha^{n-1}+\cdots+c_{n-1}\alpha+c_n\right) \\ & +\alpha\left(\alpha^{n-1}b_1+\alpha^{n-2}(b_2+c_1b_1)+\cdots+\left(b_n+\sum_{i=1}^{n-1}c_ib_{n-i}\right)\right) \\ = \; & \alpha^{n-1}b_2+\alpha^{n-2}(b_3+c_1b_2)+\cdots+\alpha\left(b_n+\sum_{i=1}^{n-2}c_ib_{n-i}\right)+c_nb_1, \end{aligned}$$

e a outra vai para

$$\alpha^{n-1}b_2+\alpha^{n-2}(b_3+c_1b_2)+\cdots+\alpha\left(b_n+\sum_{i=1}^{n-2}c_ib_{n-i}\right)+c_nb_1+1,$$

que podemos facilmente verificar que são

$$\phi(b_2,b_3,\ldots,b_n,0) \quad \text{e} \quad \phi(b_2,b_3,\ldots,b_n,1).$$

Por outro lado, não é difícil constatar que ϕ^{-1} também preserva arestas. Logo, a função ϕ é um isomorfismo de grafos. □

Está apresentado, portanto, um isomorfismo entre o grafo de De Bruijn e o grafo sobre $\mathrm{GF}(2^n)$. A seguir veremos outro isomorfismo, pois nossas construções de provas holográficas vão identificar o grafo de De Bruijn com um grafo sobre $\mathrm{GF}(2^{n/2})\times\mathrm{GF}(2^{n/2})$.

Proposição 6.3 *Seja n um inteiro par e seja α um gerador de $GF(2^{n/2})$. Seja G o grafo em $GF(2^{n/2})\times GF(2^{n/2})$ no qual o nó representado por (σ,τ) tem arestas apontando para os nós*

$$(\tau,\alpha\sigma) \quad e \quad (\tau,\alpha\sigma+1).$$

Então G é isomorfo ao grafo de De Bruijn B_n.

Prova. Pelo Lema 6.2 vemos que G é isomorfo ao grafo nas strings binárias de tamanho n no qual o nó $(b_1,\ldots,b_{n/2},b_{n/2+1},\ldots,b_n)$ tem arestas para os nós

$$(b_{n/2+1},\ldots,b_n,b_2,\ldots,b_{n/2},0) \quad \text{e} \quad (b_{n/2+1},\ldots,b_n,b_2,\ldots,b_{n/2},1).$$

Agora fazemos um entrelaçamento com os b_i's e vemos que o grafo G é isomorfo ao grafo no qual o nó $(b_1,b_{n/2+1},b_2,b_{n/2+2},\ldots,b_{n/2},b_n)$ tem arestas para os nós

$$(b_{n/2+1},b_2,b_{n/2+2},\ldots,b_{n/2},b_n,0) \quad \text{e} \quad (b_{n/2+1},b_2,b_{n/2+2},\ldots,b_{n/2},b_n,1),$$

que facilmente se pode perceber que é idêntico ao grafo B_n. □

Finalmente, apresentamos uma descrição algébrica simples do grafo de De Bruijn estendido, que é o grafo que representa a rede de roteamento para o problema de coloração.

Teorema 6.4 *Seja n um inteiro par e α um gerador de $\mathcal{F} = GF(2^{n/2})$. Seja $\mathcal{E} = \{1, \alpha, \ldots, \alpha^{5n}\}$, e $\mathcal{E}' = \{1, \alpha, \ldots, \alpha^{5n-1}\}$. Então o grafo de De Bruijn estendido com $(5n+1)2^n$ vértices é isomorfo ao grafo sobre $\mathcal{F} \times \mathcal{F} \times \mathcal{E}$, no qual cada vértice $(x, y, z) \in \mathcal{F} \times \mathcal{F} \times \mathcal{E}'$ tem arestas apontando para os vértices*

$$(y, \alpha x, \alpha z) \quad e \quad (y, \alpha x + 1, \alpha z).$$

Assim sendo, para cada vértice $(x, y, z) \in \mathcal{F} \times \mathcal{F} \times \mathcal{E}$, as coordenadas x e y indicam a linha do nó, e a coordenada z indica a sua coluna (veja novamente a Figura 5.5). Por conveniência, freqüentemente denotaremos uma tripla (x, y, z) por um vetor $\vec{x}$. Sejam as funções

$$\begin{aligned} \rho_1 : (x, y, z) &\mapsto (y, \alpha x, \alpha z) \\ \rho_2 : (x, y, z) &\mapsto (y, \alpha x + 1, \alpha z). \end{aligned}$$

Usaremos essas duas funções, $\rho_1(\vec{x})$ e $\rho_2(\vec{x})$, para denotar os vizinhos do vértice $\vec{x}$.

6.2 Aritmetização do problema

Utilizamos a descrição algébrica dos grafos de De Bruijn estendidos dada no Teorema 6.4 para construir uma versão algébrica das regras do nosso problema de coloração.

Definimos $\mathcal{F}$, $\mathcal{E}$, $\mathcal{E}'$ do mesmo modo que no Teorema 6.4. Cada nó do nosso grafo foi identificado com um elemento de $\mathcal{F} \times \mathcal{F} \times \mathcal{E}$; agora vamos interpretar uma coloração do grafo como uma função sobre esse domínio. Para esse fim, escolhemos um subconjunto de $\mathcal{F}$, digamos $\mathcal{C} = \{c_1, \ldots, c_k\} \subset \mathcal{F}$, para representar o conjunto das cores permitidas. Dessa forma, uma instância do problema de coloração pode ser vista como uma função

$$T : \mathcal{F} \times \mathcal{F} \times \mathcal{E} \to \mathcal{C}$$

que corresponde a uma primeira atribuição de cores para cada nó do grafo. Para que se possa encaixar bem com os instrumentos desenvolvidos no Cap. 4, veremos, na realidade, a instância T do problema de coloração como um polinômio de grau $(|\mathcal{F}|, |\mathcal{F}|, |\mathcal{E}|)$ sobre $\mathcal{F}$.

Similarmente, a solução do problema (segunda atribuição de cores) é dada pela função $P(x, y, z)$, que também será vista como um polinômio de grau $(|\mathcal{F}|, |\mathcal{F}|, |\mathcal{E}|)$.

Esses valores apresentados pelo provador, para a instância (T) e para a solução (P) do problema de coloração, também são chamados *candidato a teorema* e *candidato a prova*, respectivamente.

No grafo que representa a rede de roteamento, as portas são identificadas com os nós da coluna 1, o tipo de porta está atribuído por $T(x, y, 1)$, e o valor $P(x, y, 1)$ é o que a prova afirma que essa porta dá de saída.

As regras locais de coloração do grafo serão descritas por um polinômio, de grau constante, que recebe um número constante de variáveis. Seja $\vec{x} \in \mathcal{F} \times \mathcal{F} \times \mathcal{E}'$ um vértice do grafo. Assumindo que $T(\vec{x})$, $T(\rho_1(\vec{x}))$, $T(\rho_2(\vec{x}))$, $P(\vec{x})$, $P(\rho_1(\vec{x}))$, $P(\rho_2(\vec{x}))$ estejam todos em $\mathcal{C}$, podemos formar um polinômio χ de grau constante que tem a propriedade de que

$$\chi(T(\vec{x}), T(\rho_1(\vec{x})), T(\rho_2(\vec{x})), P(\vec{x}), P(\rho_1(\vec{x})), P(\rho_2(\vec{x}))) = 0$$

se e somente se as cores atribuídas por T e P a $\vec{x}$, $\rho_1(\vec{x})$ e $\rho_2(\vec{x})$ satisfazem a todas as regras de coloração. A razão porque podemos assumir que χ tem grau constante é que seu domínio está restrito apenas a pontos de $\mathcal{C}^6$, que é um conjunto finito. Como as regras de coloração são as mesmas para todos os pontos do grafo, então χ não depende de $\vec{x}$.

A fim de checar se os polinômios T e P realmente mapeiam cada nó do grafo para uma cor do conjunto $\mathcal{C}$, em vez de um elemento arbitrário de $\mathcal{F}$, vamos formar o polinômio

$$\psi(\gamma) = \prod_{c \in \mathcal{C}} (\gamma - c),$$

ou seja, um polinômio de uma variável, com grau constante, tal que $\psi(\gamma) = 0$ se e somente se $\gamma \in \mathcal{C}$. Antes de checarmos se as regras de coloração descritas por χ são satisfeitas, devemos primeiro checar se $\psi(P(\vec{x}))$ e $\psi(T(\vec{x}))$ valem zero em cada ponto $\vec{x} \in \mathcal{F} \times \mathcal{F} \times \mathcal{E}$. Desse modo evitamos que haja entre os valores de P um elemento de $\mathcal{F}$ que, embora não seja uma cor válida, faça χ tornar-se zero.

No grafo de De Bruijn estendido, para checar se as cores atribuídas à última coluna são as mesmas que as cores atribuídas à primeira coluna, basta simplesmente checar se, para todo $(x, y) \in \mathcal{F} \times \mathcal{F}$,

$$T(x, y, 1) - T(x. y, \alpha^{5n}) = 0 \quad \text{e} \quad P(x, y, 1) - P(x, y, \alpha^{5n}) = 0.$$

Podemos agora apresentar novamente o problema de coloração de grafo como um problema relativo à existência de certos polinômios. Sendo T e P polinômios de grau

$(|\mathcal{F}|, |\mathcal{F}|, |\mathcal{E}|)$, estamos vendo T como instância do problema de coloração e P como solução do problema de coloração.

Definição 6.5 *Dizemos que* P resolve T *se*

(1) *para todo* $\vec{x} \in \mathcal{F} \times \mathcal{F} \times \mathcal{E}$,
$$\psi(P(\vec{x})) = 0 \text{ e } \psi(T(\vec{x})) = 0;$$

(2) *para todo* $\vec{x} \in \mathcal{F} \times \mathcal{F} \times \mathcal{E}'$,
$$\chi(T(\vec{x}), T(\rho_1(\vec{x})), T(\rho_2(\vec{x})), P(\vec{x}), P(\rho_1(\vec{x})), P(\rho_2(\vec{x}))) = 0;$$

(3) *para todo* $(x, y) \in \mathcal{F} \times \mathcal{F}$,
$$T(x, y, 1) - T(x, y, \alpha^{5n}) = 0 \text{ e } P(x, y, 1) - P(x, y, \alpha^{5n}) = 0.$$

Para verificar se essas três condições são válidas, não pretendemos olhar os valores de T e de P em cada ponto de $\mathcal{F} \times \mathcal{F} \times \mathcal{E}$. A intenção é fornecer uma prova holográfica tal que o checador possa fazer as verificações conferindo apenas um número constante de seus segmentos.

No próximo capítulo é construída essa prova holográfica, que consiste em codificações dos polinômios T e P junto com outras informações adicionais.

Capítulo 7

A Prova Holográfica

Vamos construir uma prova holográfica para o problema de Coloração de Grafos. Essa prova terá tamanho $n \log^{O(1)} n$ e poderá ser verificada por um checador que consulta apenas um número constante de segmentos, cada um de tamanho $\sqrt{n} \log^{O(1)} n$, onde n é o tamanho da instância desse problema.

No Capítulo 5, vemos como o problema da Satisfazibilidade de Circuito pode ser transformado em um problema de Coloração de Grafos. Para este problema o Capítulo 6 mostra uma aritmetização, após a qual temos em mãos um polinômio $T(x, y, z)$, chamado instância do problema de coloração, e um polinômio $P(x, y, z)$, chamado solução do problema de coloração. Sabemos que o checador, para ficar convencido de que P é solução de T, precisa checar se são válidas as condições (1), (2) e (3) da Definição 6.5.

Porém, não desejamos que essas condições sejam verificadas em cada ponto de $\mathcal{F} \times \mathcal{F} \times \mathcal{E}$, individualmente. Em vez disso, forneceremos apresentações (Def. 4.17) de T, de P, e de outros polinômios adicionais, de modo que o checador seja capaz de fazer a checagem examinando apenas um número constante de segmentos de cada apresentação.

7.1 Escolha dos domínios

Nosso objetivo é fornecer apresentações de P e T que sejam "verificáveis", conforme conceito que vimos na Seção 4.1. Para isso escolheremos conjuntos $\mathcal{H}$ e $\mathcal{J}$ adequados, e exigiremos que as apresentações sejam dadas sobre o domínio $\mathcal{H} \times \mathcal{H} \times \mathcal{J}$.

Chamaremos N ao tamanho de uma instância do problema (descrição do circuito), a qual consiste em uma coloração do grafo de De Bruijn circular D_r. Recordando o que

foi usado na Seção 5.2, montamos a seguinte tabela:

$$\begin{aligned} m_1 &= \text{número de portas do circuito} \\ m_2 &= \text{número de entradas do circuito} \\ m &= m_1 + m_2 \\ 2^r &\geq m \quad \text{(usamos o menor } r \text{ tal que isso vale)} \\ 2^r \cdot 5r &= \text{numero de nós do grafo } D_r \\ N &= \text{tamanho da instância} = O(2^r \cdot 5r) = O(m \log m) \end{aligned}$$

Na descrição algébrica do grafo, dada no Teorema 6.4, usamos $\mathcal{F}$ isomorfo a $\mathrm{GF}(2^{r/2})$. Agora vamos escolher um corpo $\mathcal{G}$ isomorfo a $\mathrm{GF}(2^r)$, ou seja, com o quadrado do tamanho de $\mathcal{F}$. Um fato elementar da teoria dos corpos finitos é que, assim definidos, $\mathcal{F}$ é subcorpo de $\mathcal{G}$.

Usando as definições dadas naquele teorema, temos ainda conjuntos $\mathcal{E}$ e $\mathcal{E}'$ tais que

$$\begin{aligned} |\mathcal{G}| &= 2^r \\ |\mathcal{F}| &= 2^{r/2} \\ |\mathcal{E}| &= 5r \\ |\mathcal{E}'| &= 5r - 1 \\ |\mathcal{F} \times \mathcal{F} \times \mathcal{E}| &= 2^r 5r = O(m \log m) \end{aligned}$$

Feitos esses esclarecimentos, podemos partir para a escolha dos domínios. Inicialmente vamos selecionar dois conjuntos $\mathcal{I}$ e $\mathcal{K}$ com as seguintes características. Em primeiro lugar, desejamos que $\mathcal{F} \subseteq \mathcal{I} \subseteq \mathcal{G}$ e $\mathcal{E} \subseteq \mathcal{K} \subseteq \mathcal{G}$.

Em segundo lugar, para podermos usar o Algoritmo de Verificação para Três Variáveis, $\mathcal{I}$ e $\mathcal{K}$ devem ser maiores que $\mathcal{F}$ e $\mathcal{E}$ por um certo fator constante; pois, pelo Lema 4.23, tendo como graus $d_x = d_y = |\mathcal{F}| = 2^{r/2}$ e também $d_z = |\mathcal{E}| = 5r$, conseguimos um fator c constante tal que $|\mathcal{I}| > c \cdot 2^{r/2}$ e $|\mathcal{K}| > c \cdot 5r$ são suficientes.

Na próxima seção veremos que $\mathcal{I}$ e $\mathcal{K}$ devem ser suficientemente grandes para que as apresentações de $\psi(P(x,y,z))$ e $\psi(T(x,y,z))$ sobre o domínio $\mathcal{I} \times \mathcal{I} \times \mathcal{K}$ sejam checáveis e verificáveis. Como os graus destes dois polinômios são um pouco maiores que $|\mathcal{F}|$ e $|\mathcal{E}|$, será preciso contar com uma constante um pouco maior que c mencionada acima. De qualquer forma, basta $|\mathcal{I}| = O(2^{r/2})$ e $|\mathcal{K}| = O(5r)$.

A terceira característica de que precisamos é que $\mathcal{I}+1 = \mathcal{I}$, ou seja, para todo $u \in \mathcal{I}$, vale que $u + 1 \in \mathcal{I}$. Isso é importante devido à aritmetização que usamos, na qual os vizinhos de (x,y,z) são $(y, \alpha x, \alpha z)$ e $(y, \alpha x + 1, \alpha z)$, onde α é um elemento gerador de $\mathcal{G}$.

Finalmente, uma vez escolhidos $\mathcal{I}$ e $\mathcal{K}$, tendo em vista essa mesma aritmetização, tomamos $\mathcal{H} = \mathcal{I} \cup \alpha\mathcal{I}$ e $\mathcal{J} = \mathcal{K} \cup \alpha\mathcal{K}$. Portanto, todos os vizinhos dos pontos de $\mathcal{I} \times \mathcal{I} \times \mathcal{K}$ pertencem ao domínio $\mathcal{H} \times \mathcal{H} \times \mathcal{J} \subseteq \mathcal{G} \times \mathcal{G} \times \mathcal{G}$. Quanto ao tamanho, vale $|\mathcal{H} \times \mathcal{H} \times \mathcal{J}| = O(2^{r/2} \cdot 2^{r/2} \cdot 5r) = O(r2^r)$.

7.2 O oráculo

A fim de que os testes possam ser realizados nas condições de eficiência desejadas — consulta a apenas um número constante de segmentos etc. — o provador deverá, além do candidato a teorema e do candidato a prova, fornecer algumas informações adicionais. Isso é o que veremos nesta seção.

Antes relembremos a notação de vetor ($\vec{x}$), usada para indicar um elemento (x, y, z) de $\mathcal{G} \times \mathcal{G} \times \mathcal{G}$.

Sendo ψ o polinômio que aparece na condição (1) da Definição 6.5, vamos definir alguns polinômios que serão úteis, a saber:

$$\begin{aligned}
\psi_T(\vec{x}) &= \psi(T(\vec{x})) \\
\psi_P(\vec{x}) &= \psi(P(\vec{x})) \\
T_1(\vec{x}) &= T(\rho_1(\vec{x})) \\
T_2(\vec{x}) &= T(\rho_2(\vec{x})) \\
P_1(\vec{x}) &= P(\rho_1(\vec{x})) \\
P_2(\vec{x}) &= P(\rho_2(\vec{x}))
\end{aligned}$$

Dentro do algoritmo de verificação isso não é apenas notação, pois indica o modo como o valor do polinômio é obtido. Por exemplo, utilizamos "$\psi(P(\vec{x}))$" significando que o resultado é obtido em duas etapas: primeiro lemos $P(\vec{x})$ e depois calculamos ψ nesse ponto. Outras vezes utilizamos "$\psi_P(\vec{x})$", significando que o valor foi lido a partir de uma apresentação do polinômio ψ_P.

Como ψ é um polinômio de grau constante, os graus de ψ_P e de ψ_T serão maiores que os graus de P e de T por um fator constante. E o grau de ψ é constante pois é igual ao número de cores válidas que, como se vê na pág. 41, não depende do tamanho da entrada.

Agora definimos Ψ', Ψ'' e Ψ''' do seguinte modo:

$$\begin{aligned}
\Psi'(x,y,z) &= \sum_{i=1}^{2^{r/2}} \psi_P(f_i,y,z)x^{i-1} \\
\Psi''(x,y,z) &= \sum_{j=1}^{2^{r/2}} \Psi'(x,f_j,z)y^{j-1} \\
\Psi'''(x,y,z) &= \sum_{l=0}^{5r} \Psi''(x,y,\alpha^l)z^l,
\end{aligned} \tag{7.1}$$

onde $\{f_1,\ldots,f_{2^{r/2}}\}$ são os elementos de $\mathcal{F}$.

A intenção do uso desses polinômios é melhor entendida pela observação de que

$$\Psi'''(x,y,z) = \sum_{i=1}^{2^{r/2}}\sum_{j=1}^{2^{r/2}}\sum_{l=0}^{5r} \psi(P(f_i,f_j,\alpha^l))x^{i-1}y^{j-1}z^l.$$

Desse modo, Ψ''' é o polinômio nulo se e somente se $\psi(P(x,y,z))$ vale zero em todos os pontos de $\mathcal{F}\times\mathcal{F}\times\mathcal{E}$.

De maneira semelhante definimos Ψ'_T, Ψ''_T e Ψ'''_T , os polinômios análogos correspondentes a $\psi(T(x,y,z))$.

O último conjunto que vamos definir surge a partir do polinômio χ da Definição 6.5.

$$\phi(\vec{x}) = \chi(T(\vec{x}),T_1(\vec{x}),T_2(\vec{x}),P(\vec{x}),P_1(\vec{x}),P_2(\vec{x}))$$

$$\begin{aligned}
\Phi'(x,y,z) &= \sum_{i=1}^{2^{r/2}} \phi(f_i,y,z)x^{i-1} \\
\Phi''(x,y,z) &= \sum_{j=1}^{2^{r/2}} \Phi'(x,f_j,z)y^{j-1} \\
\Phi'''(x,y,z) &= \sum_{l=0}^{5r-1} \Phi''(x,y,\alpha^l)z^l
\end{aligned}$$

Aqui o domínio que define a variável z é $\mathcal{E}'$ e, portanto, a somatória de Φ''' varia de 0 a $5r-1$.

Dadas essas definições, podemos resumir o que é a prova holográfica, que deve conter codificações de todos os polinômios citados nesta seção. Em concreto, o checador exige que o provador lhe forneça apresentações no domínio $\mathcal{H}\times\mathcal{H}\times\mathcal{J}$ dos polinômios

$$\begin{array}{cccc}
T & \psi_T & \Psi' & \Psi'_T \\
P & \psi_P & \Psi'' & \Psi''_T \\
 & & \Psi''' & \Psi'''_T
\end{array}$$

e, no domínio $\mathcal{I} \times \mathcal{I} \times \mathcal{K}$, dos polinômios

$$\begin{array}{cccc} T & P & \Phi' & \phi \\ T_1 & P_1 & \Phi'' & \\ T_2 & P_2 & \Phi''' & \end{array}$$

Observamos que nas apresentações de P e T em $\mathcal{H} \times \mathcal{H} \times \mathcal{J}$, dadas anteriormente, já estão contidas sub-apresentações de T, T_1, T_2, P, P_1, P_2 em $\mathcal{I} \times \mathcal{I} \times \mathcal{K}$. Portanto, não será necessário mostrar outras apresentações para estes seis polinômios.

Desse modo temos uma prova holográfica Π, que nada mais é do que esse conjunto das quatorze apresentações citadas acima. Se esta prova Π foi construída a partir de uma prova P correta que resolve a instância T, então o algoritmo da próxima seção aceitará Π. Caso contrário, ela será rejeitada pelo algoritmo com alta probabilidade.

7.3 O algoritmo checador

Vejamos agora a descrição das ações a serem executadas pelo checador para que se possa certificar de que P realmente resolve T. O objetivo é testar se são satisfeitas as três condições da Definição 6.5.

No algoritmo, e na discussão que vemos adiante, aparecem várias constantes não especificadas. Deve estar claro que escolhas extremas para essas constantes (adequadamente grandes ou adequadamente pequenas, conforme o caso) serão suficientes para provar o teorema que é o objetivo deste capítulo. Damos a essas constantes os nomes de ε ou de c; mas isso é para simplicidade de notação pois, quando ocorrem em lugares diferentes, os valores das constantes são diferentes.

Um aspecto que devemos conhecer antes de lermos o algoritmo é que, todas as vezes que o checador recebe uma apresentação do provador, ele deve checar se a apresentação é ε-boa para algum ε muito pequeno. Se esse teste passar, o checador, daí em diante, assume que a apresentação realmente corresponde a algum polinômio e que, sempre que consultar um dos seus segmentos, ele recebe um segmento da apresentação desse polinômio. Escolhemos ε suficientemente pequeno para que, quando essa hipótese não for verdadeira, o erro seja detectado com alta probabilidade durante a checagem.

Outro aspecto que devemos ter bem claro é que há algumas conclusões a respeito das apresentações e há outras a respeito dos polinômios. Por exemplo, dada uma apresentação A para o polinômio P, uma vez verificado que A é ε-boa, o checador sabe que A é próxima à apresentação correta de *algum* polinômio P' (não necessariamente P). Depois, quando se fizerem as checagens com relação, digamos, à condição (3) da Def. 6.5,

então o checador saberá que P' satisfaz a tal condição; mas não conseguirá saber coisa alguma sobre qualquer outra parte específica que ele não tiver lido da apresentação.

Notação 7.1 *Seja Q um polinômio genérico. Chamamos $\widehat{Q}$ à apresentação (não necessariamente correta) que o provador mostra afirmando corresponder ao polinômio Q.*

Dizer que o checador lê um valor $\widehat{Q}(\vec{x})$ *significa obter esse valor a partir da tabela (i) de $\widehat{Q}$ (conforme a Def. 4.17).*

Exceções na notação acima são os polinômio P e T pois, enquanto as outras apresentações ($\widehat{\psi_P}$, $\widehat{\Phi}$ etc.) podem estar ou não formadas corretamente em função de P e de T, sabemos que estes dois polinômios *são*, por definição, o que está nas apresentações. Logo, mesmo que T não represente a instância de um circuito satisfazível, e mesmo que P seja uma prova falsa, não faz sentido dizer que $\widehat{P}$ (ou $\widehat{T}$) não seja apresentação de P (ou de T).

A seguir vemos os passos da verificação na ordem seqüencial. A discussão de correção e as explicações estão no final desta seção.

Algoritmo 7.2 Verificação da Prova Holográfica

1. *Checar se as apresentações de T e P são ε-boas*
   para alguma constante ε pequena.

2. *Checar se as apresentações $\widehat{\Psi_T}$ e $\widehat{\Psi_P}$ são ε-boas*
   para alguma constante ε pequena.

3. *Sortear um número constante de pontos $\{\vec{x}_1, \ldots, \vec{x}_c\} \subset \mathcal{H} \times \mathcal{H} \times \mathcal{J}$.*
   Para cada ponto $\vec{x}_i$
   ler $T(\vec{x}_i)$ e $P(\vec{x}_i)$;
   computar $\psi(T(\vec{x}_i))$ e $\psi(P(\vec{x}_i))$;
   checar se esses valores coincidem com $\widehat{\psi_T}(\vec{x}_i)$ e $\widehat{\psi_P}(\vec{x}_i)$.

4. *Checar se as apresentações $\widehat{\Psi'}$, $\widehat{\Psi''}$ e $\widehat{\Psi'''}$ são ε-boas*
   para alguma constante ε pequena.
   Idem para $\widehat{\Psi'_T}$, $\widehat{\Psi''_T}$ e $\widehat{\Psi'''_T}$.

5. *Sortear um número constante de pontos $\{(y_1, z_1), \ldots, (y_c, z_c)\} \subset \mathcal{H} \times \mathcal{J}$.*
   Para cada ponto (y_i, z_i)
   checar se o polinômio de uma variável em x passando por (y_i, z_i) em $\widehat{\Psi'}$
   e o correspondente polinômio em $\widehat{\psi_P}$
   possuem a relação indicada na equação (7.1).

Idem para $\widehat{\Psi'_T}$ *e* $\widehat{\psi_T}$.
Fazer operações análogas para verificar se $\widehat{\Psi''}$ *e* $\widehat{\Psi'''}$ *foram formadas corretamente.*
Idem para $\widehat{\Psi''_T}$ *e* $\widehat{\Psi'''_T}$.

6. *Sortear um número constante de pontos* $\{\vec{x}_1, \ldots, \vec{x}_c\} \subset \mathcal{F} \times \mathcal{F} \times \mathcal{E}$.
Verificar se $\widehat{\Psi'''}(\vec{x}_i) = 0$, *para cada* i.
Idem para $\widehat{\Psi'''_T}$.

7. *Checar se a apresentação* $\widehat{\phi}$ *é* ε*-boa*
para alguma constante ε *pequena.*

8. *Sortear um número constante de pontos* $\{\vec{x}_1, \ldots, \vec{x}_c\} \subset \mathcal{I} \times \mathcal{I} \times \mathcal{K}$.
Para cada ponto $\vec{x}_i$
ler $T(\vec{x}_i)$, $T_1(\vec{x}_i)$, $T_2(\vec{x}_i)$, $P(\vec{x}_i)$, $P_1(\vec{x}_i)$ *e* $P_2(\vec{x}_i)$;
checar se $\widehat{\phi}(\vec{x}_i) = \chi(T(\vec{x}_i), T_1(\vec{x}_i), T_2(\vec{x}_i), P(\vec{x}_i), P_1(\vec{x}_i), P_2(\vec{x}_i))$.

9. *Análogo ao Passo 4, para* $\widehat{\Phi'}$, $\widehat{\Phi''}$, $\widehat{\Phi'''}$.

10. *Análogo ao Passo 5, para* $\widehat{\Phi'}$, $\widehat{\Phi''}$, $\widehat{\Phi'''}$, $\widehat{\phi}$.

11. *Análogo ao Passo 6, para* $\widehat{\Phi'''}$.

12. *Checar se as sub-apresentações de* P *em* $\mathcal{H} \times \mathcal{H} \times 1$ *e* $\mathcal{H} \times \mathcal{H} \times \alpha^{5n}$
são ε*-boas para alguma constante* ε *pequena.*
Checar isso nos "mesmos pontos"
(i.é, para cada $(x_0, y_0, 1) \in \mathcal{H} \times \mathcal{H} \times 1$ *testado na primeira apresentação,*
usa-se $(x_0, y_0, \alpha^{5n}) \in \mathcal{H} \times \mathcal{H} \times \alpha^{5n}$ *para o teste da segunda).*
Certificar-se de que concordam nesses pontos.
Idem para T.

13. *Checar se os polinômios de duas variáveis*
aos quais as apresentações de P *em* $\mathcal{H} \times \mathcal{H} \times 1$ *e* $\mathcal{H} \times \mathcal{H} \times \alpha^{5n}$ *são próximas*
são as restrições do polinômio de três variáveis
ao qual a apresentação de P *é próxima,*
usando o Passo 3 do Algoritmo 4.22.
Idem para T.

Vamos às explicações sobre esse algoritmo. Nos Passos 1 e 2, o checador verifica que as apresentações de P, de T, $\widehat{\psi_P}$ e $\widehat{\psi_T}$ são ε-boas. A partir daí, vai assumir que são próximas a apresentações corretas de polinômios com os graus desejados e vai querer checar se esses polinômios são, de fato, ψ_P e ψ_T. Isso é feito no Passo 3.

Pelo Lema 4.18, sabemos que se $\widehat{\psi_P}$ e $\widehat{\psi_T}$ fossem apresentações de polinômios diferentes de ψ_P e ψ_T então o Passo 3 detectaria este fato com alta probabilidade. Portanto, se a apresentação passou por esse teste, o checador fica seguro em assumir que, sempre que lê um segmento de $\widehat{\psi_P}$ ou $\widehat{\psi_T}$, ele de fato lê um segmento das apresentações corretas de ψ_P ou ψ_T.

O próximo objetivo é checar eficientemente se os polinômios Ψ', Ψ'' e Ψ''' estão formados corretamente. Isso é feito no Passo 5. Reparemos que Ψ' depende de ψ_P, que já está verificado no Passo 3.

No Passo 5, a verificação da equação (7.1) exige calcular o polinômio de uma variável em muitos (da ordem de $2^{r/2}$) pontos do domínio. Esse cálculo pode ser feito eficientemente através de uma Transformada Finita de Fourier. Mais detalhes estão na Seção 8.3.

Esse teste é suficiente para convencer o checador de que $\widehat{\Psi'}$ foi formada corretamente porque, após o Passo 4, o checador está seguro de que a apresentação $\widehat{\Psi'}$ é uma apresentação com o grau apropriado e porque o Lema 4.18 implica que o teste no Passo 5 falhará com alta probabilidade se $\widehat{\Psi'}$ não for a apresentação do polinômio Ψ'. Similarmente, o checador está convencido de que $\widehat{\Psi''}$ e $\widehat{\Psi'''}$ são apresentações dos polinômios Ψ'' e Ψ''' e de que satisfazem as relações desejadas com ψ_P.

Após o Passo 6, aplicando novamente o Lema 4.18, o checador fica seguro de que $\Psi''' \equiv 0$.

Se a prova passou em todos os passos até agora, então o checador já pode ficar confiante em que as apresentações de P e T satisfazem a condição (1) da Def. 6.5. O próximo objetivo é verificar a condição (2) e, para tanto, os procedimentos são muito similares a esses que acabamos de observar. Portanto veremos em detalhes apenas os lugares em que diferem.

Quanto às apresentações de $T(\vec{x})$, $T_1(\vec{x})$, $T_2(\vec{x})$, $P(\vec{x})$, $P_1(\vec{x})$, $P_2(\vec{x})$, que foram dadas sobre o domínio $\mathcal{I} \times \mathcal{I} \times \mathcal{K}$, o checador assumirá que, sempre que consultar uma dessas apresentações em um ponto, ele receberá o valor do polinômio ao qual a apresentação é próxima.

Isso é assim porque, como as apresentações de T e P em $\mathcal{H} \times \mathcal{H} \times \mathcal{J}$ contêm apresentações de T, T_1, T_2, P, P_1, P_2 em $\mathcal{I} \times \mathcal{I} \times \mathcal{K}$ então, uma vez que o Passo 1 já verificou que as apresentações de T e P sobre $\mathcal{H} \times \mathcal{H} \times \mathcal{J}$ são ε-boas, concluímos que estas outras seis apresentações sobre $\mathcal{I} \times \mathcal{I} \times \mathcal{K}$ são 8ε-boas. A justificativa vem da aplicação da Proposição 4.21 pois sabemos que, por definição, $\mathcal{H} = \mathcal{I} \cup \alpha\mathcal{I}$ e $\mathcal{J} = \mathcal{K} \cup \alpha\mathcal{K}$; logo $2|\mathcal{I}| \geq |\mathcal{H}|$ e $2|\mathcal{K}| \geq |\mathcal{J}|$, e portanto temos $8|\mathcal{I} \times \mathcal{I} \times \mathcal{K}| \geq |\mathcal{H} \times \mathcal{H} \times \mathcal{J}|$, e isso satisfaz a hipótese daquela proposição.

Após o Passo 7, o checador sabe que $\widehat{\phi}$ é próxima a uma apresentação correta de algum polinômio. Então faz-se necessário verificar se esse polinômio foi formado adequadamente a partir das apresentações de T e P. Ainda outra vez usando o Lema 4.18, sabemos que se $\widehat{\phi}$ não for próxima à apresentação do polinômio desejado, isso será detectado no Passo 8 com alta probabilidade.

Agora o checador só precisa checar se ϕ é zero sobre $\mathcal{F} \times \mathcal{F} \times \mathcal{E}'$. Isso se faz exatamente do mesmo modo que para a ψ, ou seja, fazendo para o polinômio Φ''' as mesmas verificações feitas para o polinômio Ψ'''. Dessa forma, os Passos 9, 10 e 11 são perfeitamente análogos aos Passos 4, 5 e 6 que já foram comentados.

Falta só verificar se a condição (3) é satisfeita; isso é feito nos Passos 12 e 13. Como uma apresentação ε-boa é próxima de uma única apresentação correta, o fato de ambas as apresentações serem do mesmo tamanho e de que o passo 12 as testa nos "mesmos" pontos garante que sejam próximas da apresentação correta do mesmo polinômio.

Notemos que, se o checador não executasse o Passo 13, então seria possível, para as sub-apresentações de T e P em $\mathcal{H} \times \mathcal{H} \times 1$ e em $\mathcal{H} \times \mathcal{H} \times \alpha^{5n}$, que fossem idênticas entre si embora não tendo nada a ver com o resto das apresentações.

Dessa forma, fica completa a checagem da prova holográfica.

7.4 Análise da eficiência

Vamos mostrar que a prova holográfica apresentada na Seção 7.2 é maior que a instância por apenas um fator polilogarítmico e que o algoritmo construído na Seção 7.3 consulta apenas um número constante de segmentos dessa prova.

Relembremos que a prova holográfica não consiste apenas na apresentação da solução (P) do problema de coloração. Também faz parte da prova o conjunto de apresentações dos polinômios adicionais vistos na Seção 7.2. No seguinte lema analisamos o tamanho dessas codificações.

Lema 7.3 *Cada apresentação que está na prova holográfica tem tamanho* $N \log^{O(1)} N$, *onde* N *é o tamanho da instância.*

Prova. Na prova holográfica, a maior apresentação que ocorre refere-se a um polinômio de grau $(O(2^{r/2}), O(2^{r/2}), O(5r))$ e é dada sobre o domínio $\mathcal{H} \times \mathcal{H} \times \mathcal{J}$; e sabemos que $\mathcal{H}, \mathcal{J} \subseteq \mathcal{G}$. Cada apresentação é composta de quatro tabelas (Def. 4.17).

A tabela (i) possui $|\mathcal{H} \times \mathcal{H} \times \mathcal{J}| = O(2^{r/2} \cdot 2^{r/2} \cdot 5r) = O(2^r \cdot 5r)$ posições; e cada posição contém um elemento de $\mathcal{G}$, que mede r bits. Temos um total de $O(2^r \cdot 5r^2)$ bits.

A tabela (ii) possui $|\mathcal{H} \times \mathcal{H}| = O(2^{r/2} \cdot 2^{r/2}) = O(2^r)$ polinômios de uma variável; cada polinômio é descrito por $5r$ coeficientes, que é o grau na direção z; cada coeficiente é um elemento de $\mathcal{G}$, e ocupa r bits. Novamente temos $O(2^r \cdot 5r^2)$ bits.

A tabela (iii) possui o mesmo formato que a tabela (iv). Nesta última, há $|\mathcal{H} \times \mathcal{J}| = O(2^{r/2} \cdot 5r)$ polinômios de uma variável; cada polinômio é descrito por $O(2^{r/2})$ coeficientes, que é o grau na direção x; cada coeficiente ocupa r bits. Logo, há $O(2^{r/2} 5r \cdot 2^{r/2} \cdot r) = O(2^r \cdot 5r^2)$ bits.

Portanto, embora sejam de formatos diferentes, cada tabela possui o mesmo tamanho. E o tamanho total da apresentação é $O(2^r \cdot 5r^2) = O(m \log^2 m) = N \log^{O(1)} N$ bits. □

Corolário 7.4 *Para uma instância de tamanho N, a prova holográfica π construída nesta seção, possui tamanho $N \log^{O(1)} N$.*

Prova. A prova π contém apresentações dos polinômios

$$T \quad P \quad \psi_P \quad \Psi' \quad \Psi'' \quad \Psi''' \quad \psi_T \quad \Psi'_T \quad \Psi''_T \quad \Psi'''_T \quad \phi \quad \Phi' \quad \Phi'' \quad \Phi'''$$

nos domínios correspondentes. O interessante é que π é composta de um número *constante* de apresentações. □

Lema 7.5 *Na prova holográfica, os segmentos possuem tamanho $\sqrt{N} \log^{O(1)} N$, onde N é o tamanho da instância.*

Prova. Na demonstração do Lema 7.3 já falamos sobre o conteúdo de cada tabela, com os respectivos tamanhos. Os segmentos na tabela (i) são elementos de $\mathcal{G}$ e, portanto, têm r bits cada. Na tabela (ii) os segmentos são polinômios cuja descrição ocupa $O(5r^2)$ bits. Os segmentos de maior tamanho são os da tabelas (iii) e (iv) que são descrições de polinômios de grau $O(2^{r/2})$, cujos coeficientes estão em $\mathcal{G}$. Estes últimos segmentos ocupam $O(2^{r/2} \cdot r) = O(\sqrt{m} \cdot \log m) = \sqrt{N} \log^{O(1)} N$ bits. □

Teorema 7.6 *Existe um algoritmo probabilístico V que recebe como entrada uma apresentação de uma instância T de um problema de coloração, e uma prova holográfica. Sendo $|T| = n$, esse algoritmo é tal que*

(a) se a instância do problema de coloração é solúvel, então existe uma prova holográfica que faz V aceitar com probabilidade 1;

(b) se a instância não é solúvel, então qualquer prova será rejeitada por V com probabilidade $\geq 1/2$;

(c) o tamanho total da prova holográfica é no máximo $n \log^{O(1)} n$;

(d) V lê apenas um número constante de segmentos da prova holográfica, cada um de tamanho no máximo $\sqrt{n} \log^{O(1)} n$.

Prova. Trata-se do Algoritmo 7.2 e, para os itens (a) e (b) as explicações já foram dadas na seção anterior. O item (c) já foi provado: é o Corolário 7.4. Falta mostrar que a afirmação do item (d) realmente é verdadeira em cada passo do algoritmo.

Os Passos 1, 2, 4, 7, 9 e 12 testam se as apresentações são ε-boas. Esses testes usam o Algoritmo de Checagem para Três Variáveis, o qual, pelo Corolário 4.20, sabemos que lê um número constante de segmentos. Pelo Lema 7.5, sabemos que cada segmento tem tamanho $\sqrt{n} \log^{O(1)} n$.

O Passo 3 lê um número c constante de posições das apresentações de T, de P, $\widehat{\Psi_P}$ e $\widehat{\Psi_T}$. O Passo 8 é análogo. Similarmente, o Passo 6 lê um número constante de segmentos de $\widehat{\Psi'''}$ e o Passo 11, idem de $\widehat{\Phi'''}$.

O Passo 5 lê um número constante de polinômios de uma variável contidos nas apresentações de $\widehat{\Psi'}$, $\widehat{\Psi''}$, $\widehat{\Psi'''}$ e $\widehat{\psi}$; cada polinômio desses constitui *um* segmento. O Passo 10 é análogo.

Finalmente, o Passo 13 usa o terceiro passo do Algoritmo de Verificação para Três Variáveis. Tal processo implica na escolha de apenas um número constante de pontos e, para cada ponto, ler dois segmentos da apresentação de P. O processo é idêntico para T. □

Desse modo, temos provas holográficas de tamanho quase-linear checáveis pela consulta a um número constante de segmentos. No entanto, o tamanho dos segmentos (e portanto o número total de bits lidos) depende do tamanho da entrada. No próximo capítulo vemos que, após aplicar esse sistema pcp recursivamente, pode-se obter provas de tamanho quase-linear checáveis pela consulta a um número constante de bits.

Capítulo 8

Recursão

Neste capítulo mostramos como podemos aplicar recursivamente as provas holográficas construídas no Cap. 7 e assim, para qualquer $\varepsilon > 0$, podemos obter provas holográficas de tamanho $O(n^{1+\varepsilon})$ checáveis pela consulta a um número constante de bits, para instâncias de tamanho n.

8.1 Como *seria* a recursão

No Cap. 7 vemos um sistema de provas holográficas que pode ser checado pelo Algoritmo 7.2. Esse algoritmo compõe-se de várias sub-tarefas; em cada uma delas o checador lê alguns segmentos da prova e faz uma operação. Se recordarmos os Passos 1 a 13 do algoritmo, faremos a seguinte constatação.

Lema 8.1 *As operações realizadas no Algoritmo 7.2 são apenas de quatro tipos:*

- *checar se um dado polinômio vale zero em um certo ponto;*
- *checar se um dado polinômio assume um dado valor em um certo ponto;*
- *checar se algum polinômio de grau constante, calculado em um dado ponto, assume um certo valor;*
- *nos Passos 5 e 10, checar se um dado polinômio de uma variável assume um dado conjunto de valores em um conjunto de pontos, onde o tamanho desse conjunto é menor que o grau do polinômio.*

Prova. Isso é certificado por inspeção nos passos do Algoritmo 7.2, lembrando que para checar se uma apresentação é ε-boa utiliza-se o Algoritmo de Checagem para Três Va-

riáveis, do Corolário 4.20. □

A principal idéia por trás da recursão é fornecer provas holográficas de que cada uma dessas operações pode ser realizada corretamente; desse modo o checador não precisará realmente efetuar as sub-tarefas.

Vejamos uma explicação em maiores detalhes. Nosso problema é checar probabilisticamente se um circuito C é satisfazível. No início do processo o checador mostra uma instância T do problema de coloração que se refere ao circuito C. Chamemos n ao tamanho de T. Em resposta, o provador mostra uma prova holográfica Π, construída conforme vemos na Seção 7.2, e que tem tamanho $n \log^{O(1)} n$. Esse foi, chamemos assim, o "nível zero" de construção de provas holográficas.

Tal prova pode ser verificada pela máquina V do Teorema 7.6, que é probabilística. O verificador V fará sorteio de quais posições de Π serão lidas e sobre as quais executará as operações descritas no Cap. 7.

Podemos reparar que o checador não necessita ler toda a prova Π, mas consegue fazer a verificação consultando apenas um número constante de segmentos, cada um de tamanho $\sqrt{n} \log^{O(1)} n$. No entanto, queremos algo ainda mais eficiente. A tática para obter melhores checadores é a construção de provas holográficas usando o processo recursivo que descrevemos a seguir.

A máquina V é probabilística. No entanto, podemos dividir suas ações em duas partes: uma parte inicial, que sorteia todas as posições que devem ser checadas pelos Passos 1 a 13 do algoritmo; e a segunda parte que, uma vez que estas posições estão fixas, tem o comportamento totalmente definido (portanto é determinística) e vamos chamá-la de V^d.

Essa máquina V^d executa várias operações menores, que podem ser dos quatro tipos listados no Lema 8.1. Vamos chamar de μ o número de operações, o qual é constante pois é constante o número de passos do algoritmo e é constante o número de repetições de cada *loop*.

Mais adiante vemos que esses procedimentos de V^d podem ser simulados por circuitos booleanos $C_1, C_2, \ldots, C_\mu$. Assim sendo, a máquina V^d aceita se e somente se, para $i = 1, \ldots, \mu$, cada circuito C_i aceita a parte de Π que lhe cabe, por sorteio, verificar.

O truque é o seguinte. Em vez de executar a máquina V^d, o checador vai checar se os circuitos C_i são satisfeitos pelas entradas em questão. Porém, a checagem dos C_i é probabilística, isto é, o provador deve apresentar uma prova holográfica Π_1, que contém respostas às instâncias $T_1, T_2, \ldots, T_\mu$ dos problemas de coloração correspondentes aos

C_i. E é isso que denominamos recursão! Começamos com um circuito C e obtivemos

$$C_1,\ C_2,\ \ldots,\ C_\mu,$$

que devem ser checados probabilisticamente. A construção de Π_1 será denominada "nível 1 da recursão".

Chamamos Π_1, Π_2, Π_3 etc. ao conjunto de provas construídas a cada nível.

Seguindo adiante, em vez de checar se um circuito C_i é satisfazível, o checador espera encontrar (em Π_2) provas holográficas $\pi_{i1}, \pi_{i2}, \cdots, \pi_{i\mu}$ de que as μ operações da verificação de C_i podem ser realizadas corretamente. Assim, após o nível 2, o checador deseja verificar μ^2 circuitos, a saber,

$$\begin{array}{llll} C_{11}, & C_{12}, & \ldots, & C_{1\mu}, \\ C_{21}, & C_{22}, & \ldots, & C_{2\mu}, \\ \vdots & & & \vdots \\ C_{\mu 1}, & C_{\mu 2}, & \ldots, & C_{\mu\mu}. \end{array}$$

Generalizando, vemos que, após o nível k da recursão, o checador está com μ^k circuitos que devem ser checados probabilisticamente. No nível $k+1$, para cada circuito C_σ (nesta notação, σ é uma string de comprimento $|\sigma| = k$) o checador, em vez de verificar a prova, vai esperar provas holográficas para os circuitos $C_{\sigma 1}, C_{\sigma 2}, \ldots, C_{\sigma\mu}$ relativos a cada uma das μ operações da verificação.

Na Seção 8.6 vemos como finalizar esse processo recursivo.

8.2 Entradas codificadas

Ao usar recursivamente o sistema de provas do Cap. 7, há um aspecto a que devemos prestar mais atenção. Quando o checador verifica que uma prova está correta, fica convencido de que existe uma atribuição que satisfaz o circuito apresentado, mas ainda praticamente não aprendeu coisa alguma sobre essa atribuição.

Explicando melhor: vimos que uma operação feita sobre a entrada pode ser simulada por um circuito C. O provador mostra uma prova holográfica de que esse circuito é satisfazível. Quando o checador verifica essa prova, usa o Algoritmo 7.2 e fica convencido de que existe uma atribuição que satisfaz C. Então surge a questão: qual é essa atribuição?

No nível zero, isso não importa, pois basta que o provador mostre que há *alguma* atribuição. Porém, para cada nível $j \geq 1$, para os circuitos dentro de Π_j, não basta apenas checar que o circuito é satisfazível; é preciso verificar que é satisfeito *pelas* entradas desejadas e não por outras.

Por exemplo, seja a seguinte operação feita na máquina V^d.

Verificar se o polinômio $p(x)$
assume o valor v no ponto x_0.

Considere o circuito C_{x_0} que recebe como entradas um polinômio e um valor, e aceita se e somente se esse polinômio tem aquele valor quando calculado em x_0. Repare que a prova holográfica para a operação acima é exatamente uma prova π_{x_0} de que C_{x_0} é satisfazível. Logo, se não for certificado que a prova foi montada a partir de $p(x)$ e de v, pode acontecer que o checador aceite uma prova holográfica π' que mostre que C_{x_0} é satisfeita por um polinômio $p'(x)$ e um valor v' tais que $p'(x_0) = v'$ mas $p' \neq p$ ou $v' \neq v$. E não desejamos isso.

Portanto, precisamos de algum modo de checar se uma parte de uma atribuição satisfatória em uma prova holográfica é a mesma que um pedaço do dado externo. O ponto crucial é que precisamos fazer isso sem ler mais do que um número constante de bits.

A solução que adotamos é a mesma utilizada em vários artigos [BFLS91, AS92, ALM+92]: a apresentação codificada da própria instância do problema! Para isso escolhemos um código de correção-de-erro tal que o checador só precise ler um número constante de bits da string codificada. Vamos usar um código superconcentrador E.

Vamos fazer as seguintes alterações no algoritmo que tínhamos visto. Em primeiro lugar, em vez de escrever a descrição de $p(x)$ e de v do jeito normal, vamos pedir que eles sejam apresentados como $E(p(x))$ e $E(v)$. Em segundo lugar, modificamos o circuito C_{x_0} e produzimos um circuito C' que espera que suas entradas sejam codificadas através do código E. O circuito C' aceitará se e somente se suas entradas forem palavras-código de E que codificam um polinômio e um valor que fariam C_{x_0} aceitar.

Nosso checador vai pedir uma prova holográfica de que C' aceita nas entradas $E(p(x))$ e $E(v)$. Após fazer a verificação, para ficar convencido de que a atribuição satisfatória na prova — que se presume ser $E(p(x))$ e $E(v)$ — é a mesma que o dado externo, o checador vai ler alguns bits sorteados do dado externo e checar se eles coincidem com as entradas correspondentes na prova apresentada. E isso pode ser feito porque as apresentações na prova são verificáveis e, portanto, o checador pode verificar o valor de qualquer porta no circuito codificado na prova holográfica.

O objetivo é que seja constante o número de comparações entre bits da prova e do dado externo. Isso é possível porque é constante a distância mínima relativa do código usado (pois E é superconcentrador); portanto o checador, após comparar um número constante de bits, fica convencido de que o dado externo é próximo a uma única palavra-código de E, e de que essa palavra-código é a mesma que está na atribuição satisfatória

da prova holográfica.

Para analisar o tamanho do circuito obtido, basta ver que C' pode ser construído fazendo as entradas passarem por decodificadores; e as saídas destes serviriam de entrada para o próprio C (veja Fig. 8.1). O tamanho total desses codificadores é linear nas entradas, pois o código E é superconcentrador e possui algoritmo de tempo linear para decodificação. Portanto, o tamanho de C' excede o tamanho de C por no máximo uma constante vezes o tamanho das entradas.

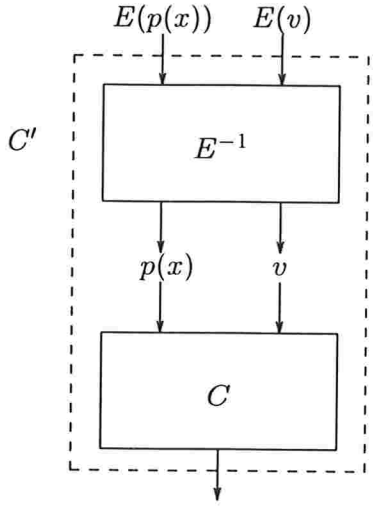

Figura 8.1: Circuito com entradas codificadas

8.3 Uso da Transformada de Fourier

Vamos construir provas holográficas para todas as operações que o checador possa querer verificar. E gostaríamos que o tamanho total dessas provas não fosse grande demais.

Vejamos os polinômios de uma variável dentro da apresentação (de três variáveis). Estes são dados através dos seus coeficientes, e o checador pode querer calcular qualquer desses polinômios em qualquer dos pontos do domínio. O modo mais natural de fazer as construções é fornecer provas holográficas do tipo "tal polinômio assume tal valor em tal ponto"; mas teríamos que fazer isso para cada polinômio em cada ponto; desse modo criaríamos provas muito grandes, conforme explicado a seguir.

A prova holográfica de que um polinômio de grau g assume um certo valor em um certo ponto é de tamanho pelo menos $\Theta(g)$ bits. Na nossa representação (tomemos uma direção, por exemplo, y) há $\Theta(2^{r/2}r)$ polinômios de uma variável, de grau $\Theta(2^{r/2})$

em y, e devem ser calculados em $|\mathcal{H}| = \Theta(2^{r/2})$ pontos. Logo, essas provas, juntas, possuem tamanho $\Theta(2^{3r/2}r) = \Theta(m^{3/2} \log m)$ bits, e isso é maior do que o quase-linear que buscamos.

A fim de prevenir esse estouro no tamanho, combinamos muitas provas em uma. Para cada polinômio, criamos uma única prova holográfica que fornece os valores desse polinômio em todos os pontos em que estamos interessados; isto é, na direção x (idem na y), os pontos do conjunto $\mathcal{H}$ e, na direção z, os pontos do conjunto $\mathcal{J}$. Sabemos que $|\mathcal{H}| = O(2^{r/2})$ e $|\mathcal{J}| = O(5r)$, portanto os conjuntos são de tamanho maior que os graus dos polinômios de uma variável por apenas um fator constante, e podemos usar a seguinte asserção.

Lema 8.2 *Polinômios de uma variável de grau d podem ser calculados em conjuntos de tamanho $n = O(d)$ de pontos em um corpo por circuitos aritméticos de tamanho $n \log^{O(1)} n$.*

Prova. [Esboço] Um polinômio de grau d pode ser calculado eficientemente em conjuntos de d pontos, usando poucas Transformadas Discretas de Fourier; isso é mostrado em [AHU74, Seção 8.5].

Para computar essa Tranformada de Fourier podemos usar a implementação eficiente apresentada por Preparata e Sarwate [PS77]. Interessa-nos não tanto o teorema principal desse artigo, mas sim um dos seus casos particulares — o item (ii) — que trata dos corpos finitos GF(2^M).

Usando essas técnicas obtemos um circuito de tamanho $d \log^{O(1)} d$, que calcula o polinômio de grau d em uma seqüência de d pontos. No nosso caso temos $n = O(d)$ e o conjunto é maior que o grau do polinômio por apenas um fator constante. Logo, podemos compor um número constante de sub-circuitos, cada qual aplicável a um sub-conjunto (de tamanho d) de pontos, e obtemos um circuito de tamanho $n \log^{O(1)} n$ para o conjunto inteiro. □

Temos portanto um circuito de tamanho $n \log^{O(1)} n$; chamemo-lo de C^*; ele recebe como entrada um polinômio $p(x)$ e n pontos $x_1, \ldots, x_n$ e dá como saída os valores $v_1, \ldots, v_n$ tais que $v_i = p(x_i)$ para $i = 1, \ldots, n$. Veja Figura 8.2.

Para encaixar isso na nossa recursão, modificamos C^* e obtemos um circuito C booleano. Este circuito recebe como entrada o polinômio $p(x)$ e os valores $v_1, \ldots, v_n$ e aceita se e somente se $p(x_i) = v_i$ para $i = 1, \ldots, n$; onde os x_i são pontos "fixos" para o circuito.

Agora vamos usar as entradas codificadas, conforme explicado na Seção 8.2. Chamemos de C' o circuito que espera entradas codificadas no código E que codificam um

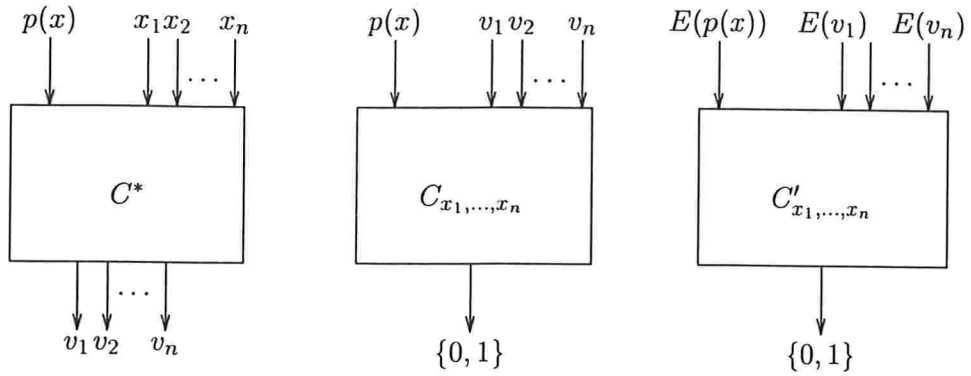

Figura 8.2: Os circuitos C^*, C e C'

polinômio e valores que fariam C aceitar.

A prova que procuramos é obtida justamente a partir de uma atribuição que satisfaz o circuito C'. Ou seja, o provador afirma que conhece os valores $v_1, \ldots, v_n$ assumidos por $p(x)$ em um domínio de tamanho n, e mostra uma prova π de que o circuito é satisfeito pelas entradas $\{E(p(x)); E(v_1), \ldots, E(v_n)\}$.

Vejamos o tamanho dessa prova π. Pelo Lema 8.2, C^* tem tamanho $n \log^{O(1)} n$. O circuito C é praticamente igual ao C^* e seu tamanho também é da ordem de $n \log^{O(1)} n$. Na Seção 8.2 vimos que a construção de C' provoca um aumento de tamanho de uma constante vezes o tamanho das entradas, ou seja,

$$|C'| \leq |C| + kn = n \log^{O(1)} n + kn = n \log^{O(1)} n.$$

Assim sendo, pelo Teorema 7.6, o tamanho da prova é

$$|\pi| = |C'| \log^{O(1)} |C'| = n \log^{O(1)} n \cdot \log^{O(1)}(n \log^{O(1)} n) = n \log^{O(1)} n.$$

Sintetizando, nesta seção vimos que, para cada polinômio de uma variável na apresentação, o provador faz todas as computações sobre um domínio de tamanho n (onde n é da ordem do grau do polinômio) e as apresenta codificadas em uma prova holográfica de tamanho $n \log^{O(1)} n$.

8.4 Como é a recursão

Vejamos explicações mais concretas sobre a recursão. A primeira seção deste capítulo serve como uma descrição geral.

Como vimos no Teorema 7.6, o nível zero constrói uma prova Π a partir de um circuito C inicial. No processo de verificação de Π surgem os circuitos $C_1, \ldots, C_\mu$. Para

a checagem destes últimos circuitos, aplicamos novamente o Teorema 7.6 (nível 1 da recursão) e obtemos a prova Π_1.

Antes de prosseguir, um aspecto a ser esclarecido é que, ao construir Π_1, o provador não pode conhecer os bits sorteados pois a prova que estamos construindo segue o esquema PCP e, por isso, não admite interação com o checador. A prova holográfica deve, portanto, ser apresentada antecipadamente, e conter as informações necessárias a todas (!) as possíveis checagens; ou seja, todas as possibilidades de sorteio.

Assim sendo, as instâncias $T_1, T_2, \ldots, T_\mu$ não podem ser interpretadas como algo que o checador deve apresentar. O que ocorre é que, após o sorteio, o checador tem em mãos essas instâncias e busca as correspondentes provas holográficas entre todas as que, antes do sorteio, foram mostradas pelo provador.

O Cap. 7 descreve a construção de uma prova holográfica que consiste em um conjunto de apresentações de polinômios em certos domínios. Para nossa recursão, cada apresentação é substituída por um conjunto de dados que chamamos *reapresentação*, conforme a definição seguinte.

Definição 8.3 *Uma* reapresentação *de um polinômio p de grau (d_x, d_y, d_z) sobre o domínio $X \times Y \times Z$ consiste em*

(a) para cada ponto no domínio, uma codificação do valor de p nesse ponto;

(b) para cada polinômio de uma variável na apresentação, uma codificação da descrição desse polinômio;

(c) para cada polinômio de uma variável na apresentação, uma prova holográfica que apresenta seus valores para cada ponto do seu domínio (por exemplo, para $x_0 \in X$ e $z_0 \in Z$ fixos, deve haver uma prova holográfica que apresenta os valores de $p|_{(x_0,\cdot,z_0)}$ em todos os pontos de Y). Essa prova deve ser verificável de modo que possa ser usada como prova de que qualquer um dos polinômios codificados assume um dos valores codificados em um ponto.

Note que o item (c) afirma que as provas holográficas apresentadas correspondem, não às instâncias T_i já citadas (que se referem a circuitos que simulam *uma* operação sobre um polinômio), mas sim a instâncias dos circuitos construídos na Seção 8.3, de modo que, para cada um dos quatro tipos de operações listados no Lema 8.1, a reapresentação contém provas holográficas de que essas operações foram realizadas com sucesso, inclusive para a operação mais difícil (feita nos Passos 5 e 10) que olha os valores de um polinômio em todo o domínio. Isso está explicado na Seção 8.3.

Na Figura 8.3 temos uma apresentação e a correspondente reapresentação. Para melhor visualização, estão representadas somente duas dimensões. A indicação "Prova: $(7x-1)(0,1,2)(-1,6,13)$" significa uma prova holográfica de que o polinômio $7x-1$ no domínio $(0,1,2)$ assume os valores $(-1,6,13)$.

	$1-y$	$2+2y$	$3+5y$	
2	-1	6	13	$7x-1$
1	0	4	8	$4x$
0	1	2	3	$x+1$
	0	1	2	

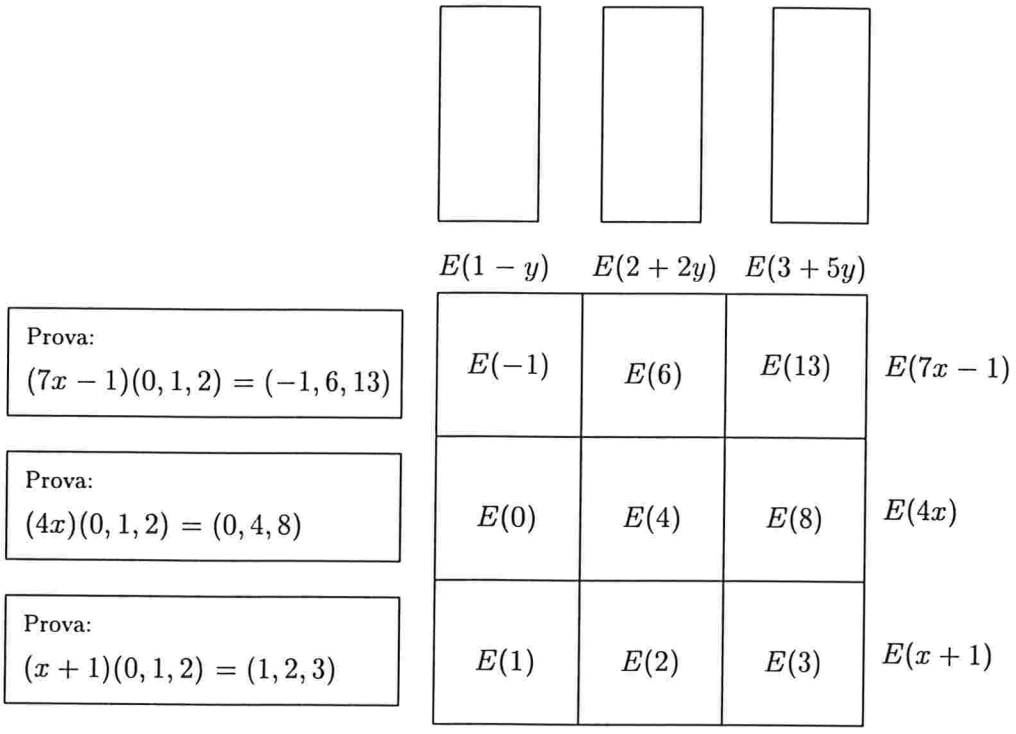

Figura 8.3: Uma apresentação e sua reapresentação

O nível 1 da recursão consiste em substituir as apresentações que estão em Π por reapresentações. Chamamos Π_1 a esse conjunto de reapresentações obtido.

Portanto, o checador não faz as μ operações de verificação das apresentações. Em lugar disto, recebe reapresentações e busca nelas as provas holográficas de que as μ operações são feitas com sucesso.

Aqui entram as técnicas explicadas na Seção 8.2. Para $i = 1, \ldots, \mu$, as checagens devem ser feitas em duas fases: primeiro, verificar a *provinha* π_i; depois, ver alguns dos bits dessa provinha e conferir se coincidem com o que deve ser sua entrada. Isto é necessário porque tais provinhas, construídas na Seção 8.3, são a combinação de muitas provas em uma; por isso precisamos ver se, contida nela, encontramos a afirmação desejada.

Por exemplo, queremos checar se $p(x) = v_j$ em x_j. Seja o circuito C construído para a fórmula booleana "$(p(x_1) = v_1) \wedge \cdots \wedge (p(x_n) = v_n)$" onde as entradas são os valores $v_1, \ldots, v_n$ e o polinômio $p(x)$. Temos uma prova π de que C é satisfazível. Após checar que π está correta, é preciso olhar se os seus bits correspondentes ao polinômio de entrada são iguais aos bits de $E(p(x))$ e também é preciso comparar os bits do j-ésimo valor de entrada com os bits de $E(v_j)$.

Com isso descrevemos de que modo é feita a checagem da reapresentação. Ou melhor, de que modo "seria feita" se o processo recursivo parasse por aqui; pois essa descrição foi apenas para o nível 1 da recursão.

Do nível 2 em diante a recursão é análoga: avançar do nível j para o nível $j + 1$ significa substituir as apresentações contidas em Π_j por reapresentações correspondentes, as quais, juntas, formam a prova holográfica que chamamos de Π_{j+1}. Na verificação, em lugar de fazer as checagens das apresentações em Π_j, o checador vai às reapresentações de Π_{j+1} buscar provas holográficas de que as tais checagens são feitas corretamente.

Daqui em diante, denotaremos por "q" o número total de níveis da recursão. O checador, conforme dito à página 54, espera uma prova final de tamanho $O(n^{1+\varepsilon})$. No fim deste capítulo vemos que o número q de níveis da recursão é escolhido em função desse ε que o checador exigir.

8.5 Tamanho total da prova

Vamos mostrar que, após q níveis da recursão, obtemos uma prova holográfica cujo tamanho total é $m(\log m)^{O(q)}$ onde m é o tamanho da instância e $q = q(\varepsilon)$ é uma constante em relação a m. Na verdade esta demonstração também vale se adotarmos qualquer função $q(m) = O(\log \log m)$; no entanto, para o teorema final na Seção 8.7, é preciso $q = O(1)$ a fim de que o número total de bits lidos seja, no máximo, uma

constante.

O seguinte símbolo nos será útil.

Notação 8.4 *Usamos o sinal "$\asymp$" para indicar que duas funções possuem igual comportamento assintótico, ou seja,*

$$f(n) \asymp g(n) \quad \textit{se e somente se} \quad f(n) = \Theta(g(n)).$$

[A] Ao avançarmos um nível na recursão, o crescimento no tamanho da prova ocorre pela substituição de uma apresentação por uma reapresentação. Na Definição 8.3 vemos que as tabelas (a) e (b) da reapresentação são codificações dos elementos presentes na apresentação. O aumento de tamanho é de apenas um fator constante, pois usamos um código E superconcentrador.

A tabela (c) é que causa a maior diferença de tamanho e a partir de agora colocamos nela a nossa atenção.

[B] Essa tabela é composta de provas holográficas para polinômios de uma variável. Nos primeiros q níveis da recursão as provas são construídas conforme explicado na Seção 8.3, e usando as técnicas do Capítulo 7. Por isso, se π é uma das provas contidas na tabela (c), sabemos que existe r tal que as (quatorze) apresentações dentro de π são, cada uma, de tamanho $\Theta(2^r r^2)$ e referem-se a polinômios de graus $(d_x, d_y, d_z) = (\Theta(2^{r/2}), \Theta(2^{r/2}), \Theta(r))$ sobre domínios $X \times Y \times Z$ tais que $|X| = |Y| = \Theta(2^{r/2})$ e $|Z| = \Theta(r)$. Os coeficientes dos polinômios estão no corpo $\mathrm{GF}(2^r)$.

Recordemos que essas apresentações são compostas por quatro tabelas; a tabela (i) tem elementos de $\mathrm{GF}(2^r)$ e as tabelas (ii), (iii) e (iv) possuem descrições dos polinômios de uma variável nas direções x, y e z, respectivamente. Na demonstração do Lema 7.3 vemos que cada uma das tabelas tem tamanho $\Theta(2^r r^2)$.

[C] Em cada nível da recursão, substituímos polinômios de uma variável por provas holográficas contendo as computações referentes a esse polinômio. Na Seção 8.3, constatamos que para um polinômio de grau d sobre domínio com $O(d)$ pontos, corresponde uma prova holográfica de tamanho $d \log^{O(1)} d$.

Chamemos (II), (III) e (IV) às tabelas no item (c) da reapresentação que correspondem, respectivamente, às tabelas (ii), (iii) e (iv) da apresentação. Seja T_x o tamanho total da tabela (II) e seja t_x o tamanho de cada prova nessa tabela. Tais provas correspondem a polinômios de grau $d_x = \Theta(2^{r/2}) = \Theta(|X|)$, portanto

$$t_x = d_x \log^{O(1)} d_x \asymp |X| (\log 2^{r/2})^{O(1)}.$$

Para o tamanho total obtemos

$$T_x = |Y \times Z| \cdot t_x \asymp |Y| \cdot |Z| \cdot |X| \cdot (\log 2^{r/2})^{O(1)}.$$

Analogamente, definimos T_y e T_z os tamanhos das tabelas (III) e (IV), e obtemos

$$T_z \asymp |X| \cdot |Y| \cdot |Z| \cdot (\log r)^{O(1)}.$$

Em conseqüência vemos que o tamanho das tabelas referentes ao eixo z são de ordem de grandeza menor que as referentes ao eixo x.

Para concluir, basta observar que $T_y = T_x$ e podemos dizer que o tamanho total da tabela (c) da reapresentação é no máximo o triplo do tamanho obtido analisando as provas referentes à direção x.

[D] Devido ao que observamos em [A], [B] e [C], vamos agora nos preocupar apenas com os tamanhos das apresentações dos polinômios na direção x. O tamanho total da prova holográfica é da mesma ordem de grandeza.

A cada nível da recursão, os polinômios de grau d são substituídos por provas holográficas de um certo tamanho $\tau = d \log^{O(1)} d$, segundo explicado na Seção 8.3. Seja k uma constante tal que $\tau = O(d \log^k d)$. As provas são construídas pela técnica do Cap. 7. Portanto, se

$$\begin{aligned} n &= \text{grau do polinômio do nível anterior, e} \\ N &= \text{tamanho da prova holográfica} \end{aligned}$$

então temos $N = O(n \log^k n) = O(2^r r^2)$ para algum r. Este $r = r_j$ é o parâmetro utilizado na escolha do corpo $\mathcal{G}_j = \mathrm{GF}(2^r)$, que compõe o domínio do nível j. Ao longo da recursão, os tamanhos das instâncias (portanto também os parâmetros $r_1, r_2, \ldots$) vão diminuindo. Logo, o corpo $\mathcal{G}_j$ é menor que o corpo usado no nível $j-1$. O tamanho da instância está amarrado com o grau do polinômio do nível anterior, e é com relação a esse grau que vamos fazer uma indução.

Voltemos a falar sobre as provas. Cada uma delas tem a seguinte estrutura:

$$\begin{array}{lllll} \text{número de polinômios} & = 2^{r/2} r \asymp \sqrt{N} & \asymp \sqrt{n}(\log n)^{k/2} & (8.1) \\ \text{tamanho dos coeficientes} & = r \asymp \log N & \asymp \log n & (8.2) \\ \text{grau do polinômio} & = 2^{r/2} \asymp \dfrac{\sqrt{N}}{\log N} & \asymp \sqrt{n}(\log n)^{(k/2)-1} & (8.3) \end{array}$$

onde, em (8.2), a medida do coeficiente é dada em bits. Podemos conferir que o tamanho total — produto desses três fatores — resulta $O(n \log^k n)$, o que está de acordo com o resultado da Seção 8.3 citado há pouco.

[E] Seja n_i uma seqüência de números tais que

$$\begin{aligned} n_0 &= n; \\ n_{j+1} &= \sqrt{n_j}(\log n_j)^{k/2}, \text{ para todo } j \geq 1. \end{aligned}$$

Então temos

$$\begin{aligned} n_1 &= \sqrt{n}(\log n)^{k/2}; \\ n_2 &= \sqrt{\sqrt{n}(\log n)^{k/2}[\log(\sqrt{n}(\log n)^{k/2})]^{k/2}} \\ &\asymp \sqrt[4]{n}(\log n)^{k/4}(\log n)^{k/2} \\ &= \sqrt[4]{n}(\log n)^{k(\frac{1}{2}+\frac{1}{4})}; \\ n_3 &\asymp \sqrt[8]{n}(\log n)^{k(\frac{1}{2}+\frac{1}{4}+\frac{1}{8})}; \end{aligned}$$

por indução, vem

$$n_j \asymp \sqrt[2^j]{n}(\log n)^{k\left(\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots+\frac{1}{2^j}\right)}.$$

[F] Voltemos agora às equações vistas em [D]. Para facilitar, adotamos

$$\text{grau} \leq \sqrt{n}(\log n)^{k/2}$$

no lugar da equação (8.3). Usando a indução em [E], obtemos o grau dos polinômios em cada nível da recursão. Para um circuito de tamanho m, com base nas fórmulas em (8.1) e (8.2), montamos a Tabela 8.1. O tamanho total é obtido pela multiplicação dos fatores referenciados nessas três equações.

j	número total de polinômios	grau do polinômio	bits	tam. total
0	$2^{r/2}r$	$2^{r/2}$	r	$2^r r^2$
0	$\sqrt{m}\log m$	$\sqrt{m}$	$\log m$	$m(\log m)^2$
1	$\sqrt{m}\log m\sqrt[4]{m}(\log m)^{k/2}$	$\sqrt[4]{m}(\log m)^{k/2}$	$\log m$	$m(\log m)^{k+2}$
2	$\sqrt{m}\log m\sqrt[4]{m}(\log m)^{k/2}$ $\times\sqrt[8]{m}(\log m)^{k(\frac{1}{2}+\frac{1}{4})}$	$\sqrt[8]{m}(\log m)^{k(\frac{1}{2}+\frac{1}{4})}$	$\log m$	$m(\log m)^{2k+2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
q		$\sqrt[2^{q+1}]{m}(\log m)^{k(\frac{1}{2}+\frac{1}{4}+\cdots+\frac{1}{2^q})}$	$\log m$	$m(\log m)^{qk+2}$

Tabela 8.1: Sobre os polinômios de uma variável

Pela tabela, portanto, vemos que, após q níveis da recursão, o tamanho total da prova é $m(\log m)^{O(q)}$ conforme enunciado no início desta seção.

Relembremos que, quanto aos fatores constantes, a cada nível precisaríamos multiplicar o tamanho por 3 (direções x, y e z) e também multiplicar por 14 (pois esse é o número de apresentações contidas dentro de cada prova holográfica).

8.6 Número de bits lidos da prova

Veremos que, a cada nível j da recursão, o número total de segmentos que o checador deve ler da prova Π_j aumenta por um fator constante; entretanto o tamanho dos segmentos diminui por aproximadamente uma raiz quadrada.

Pelo Teorema 7.6, sabemos construir uma prova holográfica Π que pode ser checada com um número constante, digamos c, de consultas a segmentos. Sobre cada um desses segmentos é feita uma operação.

Acontece que, após um nível de recursão, obtemos Π_1 onde, para cada uma dessas operações, é mostrada uma prova holográfica — que, por sua vez, pode ser checada com c consultas a segmentos. Dessa forma, em vez de consultar c segmentos de certo tamanho, é preciso consultar c^2 segmentos de tamanho menor.

Reparemos que, embora c^2 possa ser bem maior do que c, o número total de consultas feitas pelo checador ainda é um número constante. Desse modo, após q níveis da recursão, o número de consultas é c^q, mas claramente $c^q = 2^{O(q)}$.

Vejamos agora o tamanho dos segmentos. Sabemos que, dentro de uma apresentação, o maior segmento é aquele que contém a descrição de um polinômio de uma variável. A Tabela 8.1 mostra que o tamanho dessa descrição (grau vezes número de bits de cada coeficiente), diminui segundo a transformação $n \mapsto \sqrt{n}\log^{O(1)} n$. No nível q os segmentos são menores que

$$\sqrt[2^{q+1}]{m}(\log m)^k \cdot \log m.$$

Portanto, após q níveis da recursão, temos um sistema de provas que pode ser checado pela consulta de $2^{O(q)}$ segmentos, cada um de tamanho $\sqrt[2^q]{m}(\log m)^{O(1)}$.

8.7 Fim do processo recursivo

Com a finalidade de obter uma prova checável pela leitura de um número constante de bits, encerramos a recursão usando o sistema de provas de [ALM+92], tomando cuidado para não causar "estouro" no tamanho total da prova.

Nas seções anteriores observamos que, para instâncias de tamanho m, após q níveis da recursão, temos uma prova de tamanho $m(\log m)^{O(q)}$ checável pela consulta de $2^{O(q)}$ segmentos, cada um de tamanho $\sqrt[2^q]{m}(\log m)^{O(1)}$.

Relembremos que o número $q = q(\varepsilon)$ não depende de m. Na realidade, os resultados obtidos nas seções anteriores também valem para $q = q(m) = O(\log\log m)$. Porém, nesta seção, é imprescindível que q seja uma constante em relação a m.

Nos q níveis da recursão, construímos provas holográficas seguindo o esquema do Cap. 7. Porém, no nível $q+1$, para fechar o processo recursivo, usamos outra técnica: a mesma utilizada no famoso artigo [ALM+92], cuja eficiência está analisada no Corolário 3.8. Trata-se da construção das reapresentações que de fato serão verificadas pelo checador, ou seja, que estão na prova definitiva que não é mais substituída. As tabelas (a) e (b) da reapresentação permanecem com a mesma estrutura, a qual não depende do método de construção das provinhas da tabela (c).

Lema 8.5 *A prova Π_{q+1} é verificável pela consulta de um número constante de bits.*

Prova. Basta esclarecer que são checados um número constante de circuitos e que cada checagem lê um número constante de bits.

Após o nível $q+1$, há muitas provinhas contidas em Π_{q+1}. Isso ocorre porque o provador apresenta provas para todas as possibilidades de sorteio. No entanto, embora essa quantidade de provas apresentadas seja enorme, o número de circuitos que de fato são checados é apenas μ^{q+1}, constante, como se vê na Seção 8.1.

Finalmente, pelo Corolário 3.8, constatamos que a verificação lê apenas um número constante de bits de cada um dos μ^{q+1} circuitos sorteados. □

Portanto, para que seja constante o número total de bits lidos, não importa o fato de que os segmentos do nível q sejam de tamanho $\sqrt[2^q]{m}(\log m)^{O(1)}$. Tal fato é importante por outro motivo: mostrar que a aplicação do Corolário 3.8 não causa estouro no tamanho total, apesar de provocar expansão polinomial no tamanho de cada prova holográfica.

Lema 8.6 *A prova Π_{q+1} tem tamanho $m^{1+O(2^{-q})}(\log m)^{O(q)}$.*

Prova. Pelo Corolário 3.8 sabemos que, se t é o tamanho das instâncias para o nível $q+1$ da recursão, então esse nível constrói provas de tamanho $O(t^3)$.

Vejamos, primeiramente, o tamanho de cada provinha construída no nível $q+1$. Seja g o grau do polinômio do nível q. Pela Tabela 8.1 vemos que

$$g \leq \sqrt[2^q]{m}\log^k m.$$

Sabemos que a instância para um polinômio de grau g é um circuito de tamanho $O(g \log^k g)$, onde k é a constante que aparece na tabela. Desse modo, temos

$$\begin{aligned} g \log^k g &\asymp \sqrt[2^q]{m} \log^k m [\log(\sqrt[2^q]{m} \log^k m)]^k \\ &\asymp \sqrt[2^q]{m} \log^k m [\frac{1}{2^q} \log m)]^k \\ &\asymp \frac{1}{2^{qk}} \sqrt[2^q]{m} (\log m)^{2k}. \end{aligned}$$

Logo, o tamanho total da prova construída fica

$$t^3 \asymp \frac{1}{2^{3qk}} \sqrt[2^q]{m^3} (\log m)^{6k} = O(\sqrt[2^q]{m^3} (\log m)^{6k}).$$

Quanto aos polinômios em cada nível, a Tabela 8.1 mostra que o número total é

$$m^{(\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots)} (\log m)^{1+k(\frac{1}{2}+(\frac{1}{2}+\frac{1}{4})+(\frac{1}{2}+\frac{1}{4}+\frac{1}{8})+\cdots)}.$$

Logo, no nível q temos $O(m(\log m)^{kq})$ polinômios e, portanto, no nível $q+1$ temos $O(m(\log m)^{kq})$ provas construídas.

Finalmente, o tamanho total da prova é da ordem de

$$\begin{aligned} t^3 \cdot m(\log m)^{kq} &= O(\sqrt[2^q]{m^3} (\log m)^{6k} \cdot m(\log m)^{kq}) \\ &\asymp m^{1+(3/2^q)} (\log m)^{6k+kq} \\ &= m^{1+O(2^{-q})} (\log m)^{O(q)}. \end{aligned}$$

□

Lema 8.7 *Se $q = O(1)$, o número de bits randômicos utilizados na verificação é logarítmico no tamanho da entrada.*

Prova. Durante o processo recursivo não há interação entre o checador e o provador. A recursão é feita somente pelo provador que, a cada nível, procura tratar de todas as possibilidades de sorteio.

Na verificação, porém, embora deva verificar somente a prova final, o checador deve fazer sorteios relativos a cada nível de recursão, pois só assim pode saber quais posições deve ler na prova apresentada ao final da recursão.

O Algoritmo 7.2 lê apenas um número constante de segmentos de uma prova de tamanho $m \log^{O(1)} m$, onde m é o tamanho da instância. Para sortear esses segmentos bastam $\log(m \log^{O(1)} m) = O(\log m)$ bits, isto é, um número logarítmico no tamanho da entrada.

Como vimos na Seção 8.1, após o nível j da recursão, *deveriam* ser checados μ^j circuitos. Os tamanhos dos circuitos são menores a cada nível, mas mesmo que tomássemos tamanhos $O(m(\log m)^{qk+2}) = m \log^{O(1)} m$, teríamos um total de bits randômicos necessários igual a

$$\left(\sum_{j=0}^{q} \mu^j \right) \cdot O(\log m) \quad \text{bits},$$

o que ainda é um número constante vezes o fator logarítmico, pois usamos o fato de que q é uma constante.

Finalmente, no nível $q+1$, como visto na demonstração do Lema 8.6, as instâncias também são de tamanho $O(m \log^{O(1)} m)$; logo, pelo Corolário 3.8, cada checagem também usa $O(\log m)$ bits randômicos. Neste nível, o número de provas checadas é μ^{q+1}, uma constante. □

Teorema 8.8 *Para todo $\varepsilon > 0$, existe um algoritmo checador V de tempo polinomial que recebe como entrada uma codificação de uma instância T do problema de coloração de tamanho $|T| = n$ e uma prova holográfica π tais que*

- *se a instância do problema de coloração é solúvel então existe uma prova holográfica π_T que faz V aceitar com probabilidade 1;*
- *se a instância não é solúvel, então qualquer prova será rejeitada por V com probabilidade $\geq 1/2$;*
- *$|\pi| \leq n^{1+\varepsilon}$;*
- *V usa $O(\log n)$ bits aleatórios;*
- *V lê apenas um número constante de bits de T e de π.*

Prova. O algoritmo é o que está descrito neste capítulo. Aplicamos o Teorema 7.6 a si próprio q vezes. Adiante vemos como fazer a escolha de q.

Aplicar a recursão significa substituir uma apresentação pela reapresentação correspondente. Pois em vez de ser feita a checagem da apresentação recebe-se uma reapresentação, a qual contém provas holográficas de que as operações de verificação são feitas corretamente. Tais provas holográficas são verificáveis e portanto o checador pode ter certeza de que elas se referem aos objetos codificados.

Após os q níveis da recursão, obtemos uma prova Π_q que tem tamanho $n(\log n)^{O(q)}$ e pode ser checada examinando-se $2^{O(q)}$ segmentos, cada um de tamanho $\sqrt[2^q]{n}(\log n)^{O(1)}$.

Cortamos a recursão aplicando o Teorema de [ALM+92] para construir o último conjunto de provas holográficas. Com isso o número de bits lidos passa a ser constante. Por outro lado, há um aumento polinomial no tamanho de cada prova holográfica; mas isso está sob controle pois, após q níveis da recursão, as provas holográficas são tão pequenas que esse aumento não causa estouro no tamanho total (veja Lema 8.6); basta escolher q tal que

$$n^{1+O(2^{-q})}(\log n)^{O(q)} \leq n^{1+\varepsilon}$$

e, portanto, basta que o número q de níveis da recursão seja $\Theta(\log \frac{1}{\varepsilon})$. □

Dessa forma, montamos provas de tamanho quase-linear no tamanho da instância. Tal construção foi feita para um problema $\mathcal{NP}$-completo específico. Na próxima seção vemos uma discussão sobre problemas genéricos da classe $\mathcal{NP}$.

8.8 Conclusão

Dizer que um problema P pode se reduzido ao CircSat significa que existe uma transformação de tempo polinomial R que, para cada instância x de P, produz uma instância equivalente $R(x)$ do CircSat. "Equivalente" significa que a resposta (SIM ou NÃO) do CircSat com entrada $R(x)$ é a resposta correta ao problema P com instância x.

A questão é que, nessa redução, pode ser que $R(x)$ seja maior que a instância x. De fato, só podemos garantir que o tamanho da instância obtida é polinomial no tamanho da instância original. Em conseqüência, a construção de uma prova holográfica de tamanho quase-linear no $|R(x)|$ não significa obter tamanho quase-linear no $|x|$.

Então o que podemos afirmar sobre as provas holográficas construídas neste trabalho?

Tenhamos em conta uma outra definição da classe $\mathcal{NP}$, que diz que uma linguagem L pertence a $\mathcal{NP}$ se e somente se existir uma máquina de Turing M_L que, para cada inteiro n constrói, em tempo $n^{O(1)}$, um circuito booleano $C_n(x, y)$ que é tal que, para as instâncias x de tamanho $|x| = n$, vale que $x \in L$ se e somente se $C_n(x, \cdot)$ é satisfazível.

Com este ponto de vista, para cada instância x, a prova "formal" de pertinência é um par (C_x, y) onde $C_x = C_n(x, \cdot)$ e a verificação dessa prova consiste em constatar que $C_n(x, y) = 1$. Tal caracterização destaca que uma prova de pertinência a uma linguagem deve conter não apenas um certificado y mas, como é razoável, deve incluir também o conjunto das computações necessárias para verificar que esse certificado é válido (e sabemos que "avaliar um circuito" é um problema da classe $\mathcal{P}$ e, portanto, tratável). O tamanho dessa prova formal (C_x, y) é naturalmente $N = |C_x| + |y|$.

Nesse contexto, o Teorema 8.8 implica que, para *qualquer linguagem da classe* $\mathcal{NP}$, cujas provas formais de pertinência têm tamanho N, para instâncias de tamanho n, é possível construir provas holográficas de tamanho quase-linear em N. Em outras palavras, $\mathcal{NP} = PCP(\log N, 1) = PCP(\log n, 1)$, com provas de tamanho $N^{1+\varepsilon}$, para qualquer $\varepsilon > 0$.

Índice Remissivo

Referências Bibliográficas

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley series in computer science and information processing. Addison-Wesley, Reading, Massachusetts, 1974.

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. *Proof verification and hardness of approximation problems.* Proc. 33rd Ann. IEEE Symp. on Foundations of Computer Science, 1992, pp. 14–23.

[AS92] S. Arora and S. Safra. *Probabilistic checking of proofs.* Proc. 33rd Ann. IEEE Symp. on Foundations of Computer Science, 1992, pp. 2–13.

[Bab90] L. Babai. *E-mail and the unexpected power of interaction.* Proc. 5th IEEE Symp. on Structure in Complexity Theory, 1990, pp. 30–44.

[Bab94] L. Babai. *Transparent proofs and limits to approximation.* Proc. First Europ. Congr. Math., vol. 2, pp. 31–92. Birkhäuser, 1994.

[BFLS91] L. Babai, L. Fortnow, L.A. Levin, M. Szegedy. *Checking computations in polylogarithmic time.* Proc. 23rd Ann. ACM Symp. on Theory of Computing, 1991, pp. 21–31.

[Gol94] O. Goldreich. *Probabilistic proof systems (a survey).* Electronic Colloquium on Computational Complexity, 1994, Report Nr.: TR94–008.

[GMR85] S. Goldwasser, S. Micali, and G. Rackoff. *The knowledge complexity of interactive proof systems.* Proc. 17th Ann. ACM Symp. on Theory of Computing, 1985, pp. 291–304.

[GS92] P. Gemmell and M. Sudan. *Highly resilient correctors for polynomials.* Information Processing Letters 43, 1992, pp. 169–174.

[HPS94] S. Hougardy, H.J. Prömel, and A. Steger. *Probabilistically checkable proofs and their consequences for approximation algorithms.* Discrete Mathematics 136, 1994, pp. 175–223.

[KS95] Y. Kohayakawa e J.A. Soares. *Demonstrações transparentes e a impossibilidade de aproximações.* Livro do 20º Colóquio Brasileiro de Matemática, IMPA, 1995.

[Lei92] F.T. Leighton. *Introduction to parallel algorithms and architectures.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1992.

[LFKN90] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. *Algebraic methods for interactive proof systems.* Proc. 31st Ann. IEEE Symp. on Foundations of Computer Science, 1990, pp. 2–10.

[Mat97] C.A. Matsushigue. *Uma introdução técnica relativa às provas robustas checáveis probabilisticamente.* Dissertação de mestrado, IME–USP, 1997.

[Pap94] C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing Company, New York, 1994.

[PS77] F.P. Preparata and D.V. Sarwate. *Computationally complexity of Fourier transforms over finite fields.* Mathematics of Computation, vol. 31, 139, pp. 740–751, 1977.

[PS94] A. Polishchuk and D.A. Spielman. *Nearly-linear size holographic proofs.* Proc. 26th Ann. ACM Symp. on Theory of Computing, 1994, pp. 194–203.

[RS92] R. Rubinfeld and M. Sudan. *Testing polinomial functions efficiently and over rational domains.* Proc. 3rd Ann. ACM-SIAM Symp. on Discrete Algorithms, 1992, pp. 23–32.

[Sch80] J.T. Schwartz. *Fast probabilistic algorithms for verification of polynomial identities.* J. ACM 27, 1980, pp. 701–717.

[Sha90] A. Shamir. *IP = PSPACE.* Proc. 31st Ann. IEEE Symp. on Foundations of Computer Science, 1990, pp. 11–15.

[Spi95] D.A. Spielman. *Computationally efficient error-correcting codes and holographic proofs.* Ph.D. thesis, M.I.T., 1995.

[Sud92] M. Sudan. *Efficient checking of polinomials and proofs and the hardness of approximation problems.* Ph.D. thesis, U.C. Berkeley, 1992.