Léo Vieira Peres

Complexidade de circuitos booleanos

Florianópolis/SC
2017

Léo Vieira Peres

Complexidade de circuitos booleanos

Orientadora:
Prof. Dr. Jerusa Marchi

Proposta de trabalho de conclusão
de curso para a obtenção do grau de
bacharel em ciências da computação
pela Universidade Federal de Santa Catarina

Florianópolis/SC
2015

Sumário

Resumo

Podemos dizer que complexidade computacional busca descobrir o quão dificil é resolver problemas computacionais. Por exemplo, uma forma de descrever o problema em aberto mais importante da teoria da computação, $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$, é perguntar se o problema da sastifazibilidade booleana necessita de tempo "mais do que polinomial" para ser decidido no caso geral. No entanto, até agora não se obteve muito sucesso em provar limites inferiores para complexidade de problemas.

Classificar problemas pela sua complexidade de circuitos é uma duas principais frentes de pesquisa para provar limites inferior de problemas computacionais e por muitos anos pesquisadores acreditaram que complexidade de circuitos era a chave para provar problemas como $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$, onde a complexidade de circuito de um problema é basicamente o número mínimo de portas lógicas necessárias para implementar um circuito que decida este problema.

Nós veremos que as técnicas conhecidas até recentemente para provar limites inferiores são limitadas e não são suficientes para provar que $\mathsf{P} \neq \mathsf{NP}$. Entretanto, resultados recentes conseguiram se esquivar destas limitações e abriram caminho para novos tópicos de pesquisa.

A princípio, o objetivo do trabalho é realizar um estudo sobre os resultados mais recentes em complexidade de circuitos.

Palavras chaves: complexidade computacional, complexidade de circuitos.

Capítulo 1

Introdução

O objetivo central da área de complexidade computacional é saber a dificuldade intrínseca de problemas computacionais, diferentemente de design de algoritmos que busca encontrar soluções eficientes para um problema. Quando falamos de complexidade de problemas computacionais estamos querendo dizer o recurso necessário para resolver tais problemas.

Segue então que ao analisar a complexidade de problemas computacionais nós devemos levar em conta o modelo computacional e o recurso em questão. Às vezes estamos interessados em avaliar o tempo necessário para computar um certo problema em uma máquina de acesso aleatório ou talvez o número de bits que dois processadores enviam um ao outro. Alguns resultados antigos mostram que dar mais tempo ou espaço para máquinas de Turing aumenta o número de problemas que elas podem resolver, porém estes resultados não apresentaram limites inferiores para problemas naturais [AB09]. Meyer e Stockmeyer provaram que certos problemas são completos para a classe de problemas que necessitam de tempo exponencial em máquinas de Turing [For09], o que também significa que estes problemas não podem ser decididos em tempo polinomial.

Em complexidade de circuitos procura-se saber o tamanho ou profundidade dos circuitos necessários para decidir uma linguagem. Circuitos booleanos são matematicamente mais simples do que máquinas de Turing e alguns resultados em complexidade de circuitos resolveriam também problemas em aberto em outros modelos de computação. Como exemplo, se conseguirmos provar que problemas que são fáceis de se verificar uma solução não têm circuitos de tamanho polinomial ($\mathsf{NP} \not\subseteq \mathsf{P/poly}$) então $\mathsf{P} \neq \mathsf{NP}$. Porém, provar que $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ é extremamente dificil e por isso o foco de pesquisa hoje em dia é provar problemas mais fracos, na maioria das vezes restrigindo a classe de circuitos (como por exemplo, circuitos de profundidade constante). Nos anos 80 houve avanços neste sentido quando pesquisadores conseguiram mostrar que certos problemas não podem ser resolvidos por classes mais restritas de circuitos. No entanto, em 1994, Razborov e Rudich mostraram que, sob algumas hipóteses de complexidade computacional, as técnicas usadas até então para provar limites inferiores, as quais eles chamaram de provas naturais, não seriam suficiente para provar que $\mathsf{P} \neq \mathsf{NP}$ [RR94]. E por isso, para que qualquer estratégia de prova possa ser levada a frente é necessário de alguma forma passar pelas limitações das provas naturais.

Nos últimos anos, novos limites inferiores em complexidade de circuitos foram obtidos usando uma estratégia de prova que liga algoritmos "rápidos" a limite inferiores [Wil13, Wil14]. Para uma determinada classe de circuitos C, se você conseguir mostrar que o problema C–SAT (o problema de avaliar se um circuito em C não computa a função $f(x) = 0$, para todo x) tem um algoritmo mais rápido do que o algoritmo mais óbvio (tentar todas as 2^n entradas possíveis), então você consegue mostrar que a classe de problemas cuja uma solução pode ser verificada em tempo exponencial (NEXP) não tem circuitos em C. Estudar a conexão entre algoritmos e limites inferiores em circuitos booleanos é um assunto interessante para alguém que deseja realizar pesquisas em complexidade computacional.

Com o objetivo de fazer este texto autocontido no capítulo 2 nós fazemos um resumo dos fundamentos básicos necessários nos capítulos subsequentes, o que inclui algumas definições matemáticas. Em 2.3 iremos ver máquinas de Turing e algumas de suas generalizações e mostraremos que enquanto máquinas de Turing são poderosas o suficiente para capturar tudo que consideramos computável, ainda existem problemas que nos interessariam que não são computáveis por máquinas de Turing. Depois em 2.4 nós damos uma definição formal de circuitos Booleanos e definimos alguns conceitos importantes relacionados a este modelo de computação. Em 2.5 nós começamos a falar de complexidade computacional e apresentamos algumas classes de complexidade que iremos ver durante este trabalho como P, NP e PSPACE. Também apresentamos alguns resultados clássicos como alguns teoremas de hierarquia e o resultado de Baker, Gill e Solovay que prova a existência de um oráculo relativas a qual P e NP são diferentes. A seção 2.6 é sobre complexidade de circuitos e assim como vimos classes de complexidades definidas a partir de máquinas de turing nós iremos ver algumas classes de complexidades de circuitos, como elas se relacionam com outras classes de complexidades que vimos anteriormente e provaremos alguns resultados importantes que podem servir de motivação para outros resultados que irão aparecer mais para frente.

Computação relativizada e complexidade de circuitos

No capítulo 3 nós iremos ver como separações de classes de complexidade com oráculos seguem de alguns limitantes inferiores em complexidade de circuitos usando ideias que são discutidas em [Ko] e [RST15b]. Em 3.1 nós provamos de novo o teorema de Baker, Gill e Solovay desta vez usando o limitante inferior para a complexidade de consulta da função Tribes_N definida como

$$\text{Tribes}_N(x_{1,1}, x_{1,2}, \ldots, x_{2^n,n-1}, x_{2^n,n}) = \bigvee_{i=1}^{2^n} \bigwedge_{j=1}^{n} x_{i,j},$$

em que $N = n2^n$. A complexidade de consulta de uma função $\{0,1\}^n \to \{0,1\}$ é definida como a profundidade mínima entre todas as árvores de decisão que computam a função. Nós facilmente podemos provar que a função Tribes_N tem complexidade de consulta máxima N. Nos anos 80 pesquisadores estavam interessados em saber a relação entre outras classes de complexidades relativas a algum oráculo. Em especial eles queriam saber se existe algum oráculo que separa as classes PSPACE e PH e também se existe algum oráculo relativo a qual a hierarquia polinomial é infinita. Mostrando que a hierarquia polinomial pode ser expressa por circuitos de profundidade constante obtém-se que estes resultados seguem a partir de limitantes inferiores para o tamanho de circuitos de profundidade constante que computam determinadas funções, sendo estas as funções Parity_n e as funções de Sipser que iremos denotar por $f^{m,d}$. A definição destas duas funções aparece em 3.7 e 3.11 respectivamente. Nós iremos ver uma prova das seguintes implicações.

- As funções Parity_n exigem circuitos de profundidade constante de tamanho superpolinomial $\Rightarrow$ existe um oráculo A tal que $\mathsf{PH}^A \neq \mathsf{PSPACE}^A$.
- As funções de Sipser $f^{m,d}$ exigem circuitos de profundidade constante de tamanho superpolinomial $\Rightarrow$ existe um oráculo A tal que a hierarquia polinomial é infinita relativa a A.

Nós deixamos as provas dos dois limitantes inferiores acima para o capítulo 4. Ao provar cada um destes limitantes inferiores nós também obteremos que as funções Parity_n e as funções de Sipser $f^{m,d}$ não podem nem mesmo ser aproximadas por circuitos de profundidade constante e tamanho polinomial. Pela lei zero-um de Kolmogorov nós podemos ainda provar as seguintes implicações.

- As funções Parity_n não podem ser aproximadas por circuitos de profundidadade constante e tamanho polinomial $\Rightarrow \mathsf{PH}^A \neq \mathsf{PSPACE}^A$ para um oráculo aleatório com probabilidade 1.

- As funções de Sipser $f^{m,d}$ não podem ser aproximadas por circuitos de profundidade constante e tamanho polinomial $\Rightarrow$ a hierarquia polinomial é infinita relativa a um oráculo aleatório com probabilidade 1.

Dizer que, por exemplo, PSPACE e PH são diferentes relativas a um oráculo aleatório com probabilidade 1 não significa que elas são diferentes relativas a todo os oráculos, mas sim significa que o conjunto de todos os oráculos A que satisfazem $\mathsf{PH}^A = \mathsf{PSPACE}^A$ tem medida zero. De fato, veremos que para todos os resultados do capítulo 3 existe um oráculo relativa a qual as classes de complexidade em questão colapsam. A importância de um resultado destes é que se quisermos provar relações entre estas classes nós necessariamente teremos que usar métodos de provas que não relativizam, o que basicamente significa que teremos que usar um argumento que usa algo a mais do que a capacidade de uma das classes poder simular a outra classe. No fim deste capítulo nós iremos ver um argumento que duas classes de complexidades serem diferentes relativas a um oráculo aleatório nem sequer serve como evidência que estas duas classes são diferentes no mundo não relativizado [For94].

Restrições e projeções aleatórias

O capítulo 4 é inteiramente voltado para provar os limitantes inferiores enunciados no capítulo 3 (ver 3.8, 3.9, 3.12 e 3.14). Na seção 4.1 nós iremos ver uma prova que as funções Parity_n exigem circuitos de profundidade constante que tenham um número exponencial de portas lógicas. O método utilizado é o de restrições aleatórias introduzidos em [Sub61] que funciona da seguinte forma. Suponha que tenhas uma função Booleana $f : \{0,1\}^n \to \{0,1\}$ com variáveis de entrada $x_1, x_2, \ldots, x_n$. Uma restrição aleatória ρ irá atribuir valores em $\{0,1,*\}$ de forma aleatória e independente para cada uma das n variáveis de f, em que $*$ denota que a variável permanece livre. Chame de $\rho(x_i)$ o valor atribuido à i-ésima variável, então a função resultante $f_{|\rho}$ satisfaz

$$f_{|\rho}(x_1, x_2, \ldots, x_n) = 1 \iff f(\rho(x_1), \rho(x_2), \ldots, \rho(x_n)) = 1.$$

Agora, o que queremos provar é que todos circuitos de profundidade constante e tamanho subexponencial não podem computar Parity_n, para n suficientemente grande. Uma forma de fazer isto é provar os seguintes pontos separadamente.

1. Toda função computada por um circuito de profundidade constante e tamanho subexponencial colapsa para uma função extremamente simples quando atingida por uma restrição aleatória
2. As funções Parity_n permanecem complexas mesmo após serem atingidas por uma restrição aleatória.

Håstad em sua tese de doutorado [Hås87] provou o lema da troca de Håstad 4.2, que é essencial para provar o primeiro ponto acima. Nós iremos ver duas provas do lema da troca de Håstad, sendo a primeira prova a original de Håstad que usa indução. A segunda prova, que é atribuida a Razborov, aparece em [Bea94] e usa um argumento de contagem. O lema diz que circuitos de profundidade 2 simplificam quando atingidos por uma restrição aleatória. Então podemos para cada camada de um circuito de profundidade constante aplicar uma restrição aleatória que com probabilidade muito alta irá simplificar esta camada ao ponto de reduzir a profundidade do circuito em 1. Ao fim teremos transformado um circuito de profundidade $d \geq 2$ em algo como uma árvore de decisão de profundidade constante. Por outro lado, o item (2) acima é facilmente obtido observando que ao restringir as variáveis de entrada da função Parity_n nós ainda temos uma função paridade ou a sua negação sobre um número menor de variáveis mas que ainda não pode ser expressa por uma árvore de decisão de profundidade constante. Para provar o teorema 3.9 nós iremos usar o fato que restrições aleatórias preservam a distribuição uniforme.

Håstad conseguiu ainda na sua tese doutorado provar o teorema 3.12 usando restrições em bloco que atribui valores às variáveis de entrada de forma que variáveis num mesmo bloco não têm valores atribuidos de forma independente. Porém o método dele não é suficiente para provar 3.14 pois restrições em bloco não preservam

a distribuição uniforme. Em [RST15a], Rossman, Servedio e Tan conseguiram superar esta barreira para por fim provar 3.14. Na seção 4.2 iremos mostrar a prova de Rossman, Servedio e Tan dos teoremas 3.12 e 3.14 que usa projeções aleatórias que são uma generalização de restrições aleatórias. Agora ao invés de setar uma variável para uma constante ou manter elas livres, nós iremos particionar elas em blocos de forma que cada variável ou é feita constante ou é projetada para uma nova variável que é comum à todas as outras variáveis no mesmo bloco.

1.1 Objetivo

Este trabalho visa apresentar um estudo teórico acerca da área de complexidade de circuitos e suas aplicações ao estudo da complexidade computacional. Primeiramente será feito um estudo sobre complexidade computacional. Depois serão pesquisados tópicos em complexidades de circuitos com foco principal nos tópicos mais recentes e ferramentas/técnicas comumente usadas em provas em complexidade de circuitos.

Capítulo 2

Fundamentos

Neste capítulo nós introduzimos algumas convenções e conceitos fundamentais para entender este trabalho. Muitas das convenções usadas aqui são as mesmas encontradas em alguns dos principais livros de teoria da computação [AB09, Gol00, Sav98, LP97, Sip12].

Introduzimos primeiro na seção 2.1 algumas definições matemáticas que serão importante ao longo deste texto e também linguagens e como representar problemas computacionais como linguagens na seção 2.2. Depois falamos de máquinas de Turings e suas variantes em 2.3. Máquinas de Turing foram introduzidas por Alan Turing em [Tur36] e são a partir delas que iremos introduzir as principais classes de complexidade. Na seção 2.4 nós vemos circuitos Booleanos como um modelo de computação. Nas seções 2.5 e 2.6 nós discutiremos complexidade computacional pela primeira vez em algum detalhe.

2.1 Definições matemáticas

Ao longo deste trabalho nós vamos usar a notação $[n]$ quando queremos expressar o conjunto $\{1, 2, \ldots, n\}$ de todos os números naturais menores ou iguais a n. Também, sempre que tivermos uma sequência $x_1, x_2, \ldots, x_n$ nós podemos sucintamente usar a notação $\{x_i\}_{i\in[n]}$. Se f é uma função qualquer e S é um conjunto, então denotamos por $f^{-1}(S)$ o conjunto $\{x|f(x) \in S\}$. Se $S = \{y\}$ contém apenas um elemento então podemos escrever $f^{-1}(y)$ ao invés de $f^{-1}(S)$.

Ao longo do texto iremos várias vezes usar as seguintes desigualdades.

1. $(1 + x)^r \geq 1 + rx$, para todo número real $x \geq -1$ e todo número inteiro $r \geq 1$.
2. $1 + x \leq e^x$, para todo número real $x \in \mathbb{R}$.

O item (1) segue do teorema binomial que diz que $(1 + a)^r = \sum_{k=0}^{r} \binom{r}{k} x^k = 1 + ar + \frac{r(r-1)}{2}a^2 + \ldots$. Para a segunda desigualdade nós temos que os casos em que $x < -1$ e $x = 0$ são triviais. Para $x > 0$, nós podemos usar a expansão de Taylor de e^x:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots. \tag{2.1}$$

Então temos que e^x é igual a $1+x$ mais algum valor não negativo. Para $-1 \leq x < 0$ nós fazemos o seguinte. Seja $y > 0$ tal que $x = -\frac{1}{y}$. Então é verdade que

$$1 + x = \left(1 - \frac{1}{y}\right)^{-yx},$$

e argumentamos que

$$\lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^{-nx} = e^x. \tag{2.2}$$

Como o limite acima aproxima-se do limite e^x pela esquerda teremos então que $\left(1 - \frac{1}{n}\right)^{-nx} \leq e^x$, para todo $n > 0$. Em particular teremos que $1 + x = \left(1 - \frac{1}{y}\right)^{-yx} \leq e^x$. Podemos provar 2.2 usando a versão generalizado do teorema binomial que afirma que para todo $a, r \in \mathbb{R}$ é verdade que $(1 + a)^r = 1 + ra + \frac{r(r-1)}{2!}a^2 + \frac{r(r-1)(r-2)}{3!}a^3 + \ldots$. Então, fazendo $a = -\frac{1}{n}$ e $r = -nx$:

$$\lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^{-nx} = \lim_{n\to\infty} 1 + x + \frac{-nx(-nx-1)}{2!}\frac{1}{n^2} + \frac{-nx(-nx-1)(-nx-2)}{3!}\left(-\frac{1}{n^3}\right) + \ldots.$$

Para todo $k > 2$ o k-ésimo termo tende a $\frac{x^k}{k!}$ com n indo ao infinito. Portanto temos que

$$\lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^{-nx} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots,$$

que é a expansão de Taylor de e^x.

Notação assintótica

Em ciência da computação é comum ao estarmos considerando funções que crescem com algum parâmetro n apenas levarmos em conta o comportamento assintótico da função, que é basicamente o comportamento da função no limite dos grandes números. Assim sendo, nós a notação assintótica para classificar funções a partir de seu comportamento assintótico. Nós dizemos que

- $f = \mathcal{O}(g)$ se existem constantes $c, n_0 \in \mathbb{R}$ tais que para todo $n \geq n_0$, $f(n) \leq cg(n)$.
- $f = \Omega(g)$ se existem constantes $c, n_0 \in \mathbb{R}$ tais que para todo $n \geq n_0$, $f(n) \geq cg(n)$.
- $f = \Theta(g)$ se $f = \mathcal{O}(g)$ e $f = \Omega(g)$.
- $f = o(g)$ se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
- $f = \omega(g)$ se $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$.

Ao usar a notação assintótica estamos apenas pegando o termo de maior ordem de uma função, ignorando fatores constantes e termos de menor ordem. Por exemplo, se $f = 4n^2 + 31n + 4$ então dizemos apenas que $f = \mathcal{O}(n^2)$. Se quisermos ignorar fatores polilogaritmicos nós usamos $\widetilde{\mathcal{O}}$, $\widetilde{\Omega}$ e $\widetilde{\Theta}$. Por exemplo, temos que $n \log n = \widetilde{\Theta}(n)$ ao mesmo tempo que $n \log n = \omega(n)$.

Probabilidade e variáveis aleatórias

Iremos denotar por $\{0_p, 1_{1-p}\}$ e $\{0_{1-p}, 1_p\}$ a distribuição de bits aleatórios onde o bit 0 é tirado com probabilidade p e $1-p$, respectivamente. Para a distribuição uniforme podemos alternativamente usar as notações $\{0,1\}$ ou $\{0_{\frac{1}{2}}, 1_{\frac{1}{2}}\}$.

Sempre que quisermos denotar objetos aleatórios nós iremos destacar este objeto em negrito. Por exemplo, denotamos por $\boldsymbol{x}$ uma string aleatória tirada de $\{0,1\}^n$, o que pode ser denotado por $\boldsymbol{x} \sim \{0,1\}^n$.

Em geral, um espaço de probabilidade Ω é um conjunto $\{\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \dots\}$ e associamos a Ω uma distribuição de probabilidade $\mathcal{D} = \{p_1, p_2, \dots\}$ em que cada elemento $\boldsymbol{\omega}_i \in \Omega$ tem uma probabilidade p_i associada a ele e $\sum_{\boldsymbol{\omega}_i \in \Omega} p_i = 1$. Como já fizemos no parágrafo anterior, nós denotamos que $\boldsymbol{\omega} \in \Omega$ é tirada da distribuição $\mathcal{D}$ por $\boldsymbol{\omega} \sim \mathcal{D}$. Uma variável aleatória $\boldsymbol{X}$ em um espaço de probabilidade Ω é uma função $\boldsymbol{X} : \Omega \to \mathbb{R}$ e denotamos o valor esperado de $\boldsymbol{X}$ por $E_{\boldsymbol{\omega}_i \sim \Omega}[\boldsymbol{X}(\boldsymbol{\omega}_i)] = \sum_{\boldsymbol{\omega}_i} p_i \boldsymbol{X}(\boldsymbol{\omega}_i)$. Nós geralmente iremos apenas escrever $\boldsymbol{X}$ ao invés de $\boldsymbol{X}(\boldsymbol{\omega}_i)$, e algumas vezes iremos até mesmo omitir a distribuição quando o contexto for claro o suficiente.

As seguintes desigualdades serão de grande importância para nós. Seja $\boldsymbol{X}$ uma variável aleatória satisfazendo $E[\boldsymbol{X}] = \mu$. A desigualdade de Chernoff diz que

$$\Pr\left[\left|\boldsymbol{X} - \mu\right| \geq (1+\delta)\mu\right] \leq e^{-\frac{\delta^2}{2+\delta}\mu}. \tag{2.3}$$

E também, para $0 \leq \delta < 1$,

$$\Pr\left[\left|\boldsymbol{X} - \mu\right| \leq (1-\delta)\mu\right] \leq e^{-\frac{\delta^2}{2}\mu}. \tag{2.4}$$

No primeiro caso podemos relaxar a desigualdade para uma forma mais conveniente.

$$\Pr\left[\left|\boldsymbol{X} - \mu\right| \geq (1+\delta)\mu\right] \leq e^{-\frac{\delta^2}{3}\mu}, \text{ se } 0 \leq \delta \leq 1.$$

e também

$$\Pr\left[\left|\boldsymbol{X} - \mu\right| \geq (1+\delta)\mu\right] \leq e^{-\frac{\delta}{3}\mu}, \text{ se } \delta \geq 1.$$

A desigualdade de Hoeffding nos dá um limitante superior para a probabilidade que a soma de variáveis aleatórias se afastem demais de suas médias. Sejam $\boldsymbol{X}_1, \boldsymbol{X}_2, \dots, \boldsymbol{X}_n$ variáveis aleatórias tais que cada $\boldsymbol{X}_i \in [a,b]$. Então, para $t > 0$:

$$\Pr\left[\frac{1}{n}\Big(\sum_{i=1}^{n} \boldsymbol{X}_i - \mathsf{E}\big[\boldsymbol{X}_i\big]\Big) \geq t\right] \leq e^{-\frac{2nt^2}{(b-a)^2}}. \tag{2.5}$$

Uma variável aleatória indicadora $\boldsymbol{X}$ para algum evento é 1 se este evento acontece e 0 caso contrário. Uma variável aleatória indicadora para algum evento convenientemente satisfaz $E[\boldsymbol{X}] = \Pr[\boldsymbol{X} = 1] = \Pr[\text{o evento acontece}]$. Além do mais, sejam $\boldsymbol{X}_1, \boldsymbol{X}_2, \dots, \boldsymbol{X}_n$ varíaveis aleatórias indicadoras. Então, a desigualdade de Hoeffding (2.5) nos diz que

$$\Pr\left[\frac{1}{n}\Big(\sum_{i=1}^{n} \boldsymbol{X}_i - E\big[\boldsymbol{X}_i\big]\Big) \geq t\right] \leq e^{-2nt^2}. \tag{2.6}$$

2.2 Linguagens

Um *alfabeto* Σ é um conjunto finito e não vazio de símbolos como $\{0,1\}$ ou $\{a,b,c\}$. Uma *palavra* construída sobre um alfabeto Σ é uma sequência de símbolos de Σ. Como exemplo, se Σ for o alfabeto binário $\{0,1\}$, então 0011 e 0101 são palavras sobre Σ. Finalmente, denotamos por Σ^* o conjunto de todas as palavras formada por símbolos de Σ e definimos uma *linguagem* como um subconjunto qualquer de Σ^*.

Permitimos uma palavra vazia que não contém nenhum símbolo e denotamos esta palavra por ε e temos que $\varepsilon \in \Sigma^*$, para qualquer alfabeto Σ. O tamanho de uma palavra w é o número de símbolos que a compõem e é denotada por $|w|$ — desta forma $|\varepsilon| = 0$. Para representar o i-ésimo símbolo que compõe uma palavra w nós escreveremos w_i. Para algum inteiro $n \geq 0$, Σ^n denota o conjunto de todas as palavras de tamanho n sobre o alfabeto Σ.

Nós também queremos representar objetos como grafos, vetores, etc, através de palavras. Neste caso, se x é um objeto qualquer, então sua representação em binário será denotada por $\langle x \rangle$.

O que mais nos importa aqui é que podemos representar problemas computacionais através de linguagens, o que nos chamamos de problemas de decisão.

2.2.1 Problemas de decisão

Em problemas de decisão nós queremos decidir se um determinado elemento pertence a um conjunto S ou não. Como exemplo de um problema de decisão: dado um número natural p, nós queremos decidir se p é primo. Neste caso S é o conjunto de todos os números primos.

Para solucionar o problema de decisão de $S \subseteq \{0,1\}^*$ nós usamos uma função $f : \{0,1\}^* \to \{0,1\}$ tal que $S = \{x \mid f(x) = 1\}$. Chamamos f de *função característica* de S. Dessa forma, solucionar um problema de decisão é análogo a decidir se uma palavra pertence à uma linguagem, dado uma representação das instâncias do problema como strings binárias.

2.3 Máquinas de Turing

Uma visão intuitiva de uma máquina de Turing é a de um matemático que tem consigo uma folha de rascunho em que ele pode escrever os resultados parciais de sua computação e um conjunto finito de instruções que ele deve seguir. Formalmente, uma máquina de Turing é composta por três unidades:

- k fitas infinitas à direita que contêm células adjacentes e um cabeçote que em um dado momento se encontra em uma única célula de sua fita e que pode realizar as seguintes funções: a) escrever ou apagar um símbolo na célula em que ele se encontra b) se mover para uma das células adjacentes à sua célula atual;
- um registrador que guarda o estado atual da computação;
- um conjunto de instruções.

A computação inicia com os k cabeçotes na célula mais à esquerda de suas respectivas fitas e em um estado inicial que é o mesmo para todas as entradas. Daí em cada passo da computação os k cabeçotes irão ler o conteúdo atual das células em que eles se encontram e conforme o estado atual, o símbolo lido e o conjunto de instruções eles decidem se escrevem ou apagam um símbolo na sua célula atual (sendo que o símbolo escrito pode ser o mesmo que já se encontra naquela célula) e para qual direção eles irão se movimentar (ou se permanecerão na mesma célula). Após cada passo o registrador de estado passa a guardar um novo estado

(ou seja, o próximo estado da computação) que depende do estado atual e o símbolo lido pelos cabeçotes. A computação termina quando o registrador de estado guarda um estado de parada.

Das k fitas da máquina de Turing, a primeira é de somente leitura e a chamaremos de *fita de entrada*. As últimas $k-1$ fitas são de escrita e leitura e elas são chamadas de *fitas de trabalho*, sendo a última fita a *fita de saída*.

A seguir nós vemos uma definição formal de máquina de Turing.

Definição 2.1. *(Máquinas de Turing)*

Uma máquina de Turing M é uma tripla (Γ, Q, δ) onde Γ é o alfabeto de fita, Q é o conjunto de estados de M que contém o estado inicial q_0 e o estado de parada q_h e $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{E, N, D\}^k$ é a função de transição.

A função de transição é interpretada como $\delta(q, (\sigma_1, \ ..., \ \sigma_k)) = (q', (\sigma'_2, \ ..., \ \sigma'_k), z)$, $z \in \{E, N, D\}^k$, significando que quando o estado atual de M for q e os símbolos sendo lidos pelos cabeçotes das k fitas forem $\sigma_1, \ ..., \ \sigma_k$ então M muda o seu estado atual para q', muda o conteúdo das suas últimas $k-1$ fitas para $\sigma'_2, \ ..., \ \sigma'_k$ e os k cabeçotes da fita se movimentam conforme z (a i-ésima fita se move para a esquerda, permanece na mesma célula ou se move para direita se o valor de z_i for E, N ou D, respectivamente). Sempre que um cabeçote que estiver na célula mais à esquerda de sua fita tentar se mover para esquerda, este permanecerá na mesma célula.

A entrada de uma máquina de Turing é o conteúdo da fita de entrada antes do ínicio da computação. Denotamos o resultado da computação de M sobre uma entrada x por $M(x)$.

Neste trabalho vamos na maior parte das vezes assumir que $\Gamma = \{0, 1, \triangleright, \square\}$, onde $\triangleright$ é o símbolo que marca o começo das fitas e $\square$ é um símbolo que denota uma célula vazia.

Nós podemos representar cada passo da computação de uma máquina de Turing levando em conta o conteúdo atual das k fitas, as posições dos cabeçotes e o estado atual. Essa representação de um passo da computação de uma máquina de Turing é chamada de *configuração* e podemos mapear uma configuração para uma palavra em $\{0,1\}^*$. No ínicio da computação a máquina de Turing se encontra na *configuração inicial*. A sequência de todas as configurações que uma máquina de Turing entra durante a computação é chamada de *história de computação*. Essa visão dos passos da computação de uma máquina de Turing é útil quando queremos representar toda a computação de uma máquina de Turing como uma string. Sem nos preocurpamos com os detalhes de uma representação das configurações, vamos convencionar que a configuração inicial de qualquer computação terá o seguinte:

- Todas as fitas têm o símbolo $\triangleright$ em sua célula mais à esquerda;
- A primeira fita irá conter uma string $x \in \{0,1\}^*$ após a sua primeira célula;
- Todas as outras células de todas as fitas serão marcadas com $\square$.

2.3.1 Máquina de Turing universal

Note que precisamos somente da função de transição para descrever uma máquina de Turing. Dessa forma podemos representar máquinas de Turing como strings binárias e fazemos duas suposições:

- Cada string $\alpha \in \{0,1\}^*$ descreve uma máquina de Turing
- Cada máquina de Turing é descrita por infinitas strings

A primeira condição pode ser alcançada se mapearmos todas as strings que não são descrição válidas de máquinas de Turing para uma máquina canônica qualquer — como a máquina de Turing que rejeita todas as entradas. A segunda condição pode ser obtida se concatenarmos uma sequência de símbolos inúteis ao fim da descrição da máquina de Turing, isto não irá mudar o conjunto de instruções sendo representado se usarmos alguma sequência de bits para demarcar o fim da descrição.

De acordo com a nossa notação, denotaremos a string que descreve uma máquina de Turing M por $\langle M \rangle$. Se α é uma string, então denotaremos por M_α a máquina de Turing descrita por α.

Essa representação de máquinas de Turing como strings é útil quando queremos usar descrições de máquinas de Turing como entrada para uma outra máquina de Turing. O teorema a seguir nos diz que existe uma máquina de Turing capaz de simular a execução de qualquer máquina de Turing sobre uma entrada arbitrária.

Teorema 2.2. *(Máquina de Turing universal)*

Existe uma máquina de Turing $\mathcal{U}$ que ao receber $\langle \alpha, x \rangle$ em sua fita de entrada, $\mathcal{U}$ dá como saída o resultado da computação de M_α sobre a entrada x.

Demonstração. Precisamos apenas nos convencer que uma vez que podemos extrair da descrição de M_α (ou seja, a string α) o seu conjunto de estados e sua função de transição temos então toda informação necessária para simular a execução de M_α sobre a entrada x usando as fitas de trabalho de $\mathcal{U}$.

Porém, o número de fitas de $\mathcal{U}$ é finito (somente 3 fitas são necessárias), e $\mathcal{U}$ deve ser capaz de simular qualquer máquina de Turing com um número arbitrário de fitas. Se k é o número de fitas de M_α, então é possível fazer isto guardando o conteúdo de $k-1$ fitas (nós podemos usar a fita de saída de $\mathcal{U}$ para simular a fita de saída de M_α) de M_α em uma das fitas de trabalho de $\mathcal{U}$ particionando esta fita em $k-1$ espaços $E_1, E_2, \ldots, E_{k-1}$, onde cada espaços consecutivos são separados por um símbolo especial (como '#'). Sempre que a i-ésima fita de M_α precisar de mais espaço, movemos todos símbolos que aparecem após a última célula de E_i uma posição para a direita.

Dessa forma, após a simulação teremos $M_\alpha(x)$ escrito sobre a fita de saída de $\mathcal{U}$.

□

Um ponto importante sobre o resultado acima é que a simulação pode ser feita de forma eficiente. No capítulo seguinte iremos definir o que queremos dizer por eficiente e também veremos em detalhe uma máquina de Turing universal ainda mais eficiente do que a máquina de Turing esboçada na prova do teorema anterior.

2.3.2 Máquina de Turing não-determinística

Na nossa definição de máquinas de Turing acima, o próximo passo de uma máquina de Turing é definido somente pelos símbolos sendo lidos pelos seus cabeçotes de fita e o estado atual da máquina. Nós chamamos estas máquinas de Turing cujo o próximo passo é estritamento único de máquinas de Turing determinísticas. Por outro lado, uma máquina de Turing não-determinística tem sempre duas alternativas de próximos passos que ela deve decidir tomar.

Definição 2.3. *(Máquina de Turing não-determinística)*

Uma máquina de Turing não-determinística N é uma máquina de Turing convencional como definida em 2.1 mas com duas funções de transições δ_1 e δ_2. A cada passo de sua execução N deve escolher usar uma de suas duas funções.

Dizemos que N aceita a entrada x se existe pelo menos uma sequência de escolha das funções de transição tal que $N(x) = 1$.

O conjunto de linguagens decididas por máquinas de Turing não-determinística é o mesmo que o conjunto de linguagens decididas por máquinas de Turing determinística, isso segue pois podemos simular uma máquina de Turing não-determinística N por uma máquina de Turing M que tenta todas as possíveis sequência de escolhas da função de transição que N faz. Além disso, máquinas de Turing determinística são uma classe específica de máquinas de Turing não-determinísticas (onde δ_1 e δ_2 são idênticas).

2.3.3 Máquinas de Turing com oráculo

Um oráculo O para uma linguagem L é um dispositivo que recebe uma entrada x e dá como resposta 1 se $x \in L$ e 0 caso contrário. Nós não estamos preocupados com o funcionamento interno de um oráculo, nós vemos oráculos como "caixas pretas" donde nós simplesmente colocamos a entrada em um lado e recebemos a saída em outro lado. Máquinas de Turing com oráculo são máquinas de Turing convencionais que têm acesso a um oráculo.

Definição 2.4. *(Máquina de Turing com oráculo)*

Uma máquina de Turing M com acesso a um oráculo para L é uma máquina de Turing convencional com a adição de uma fita que chamaremos de fita de oráculo *e três estados $q_{consulta}$, q_{sim} e $q_{não}$. Sempre que M quiser consultar o oráculo para saber se uma string x' pertence a L ou não, M escreve x' sobre sua fita de oráculo e muda seu estado para $q_{consulta}$. Daí, o próximo estado de M será q_{sim} caso $x' \in L$, ou $q_{não}$ caso contrário.*

A partir de agora denotaremos uma máquina de Turing M com acesso a um oráculo para uma linguagem L por M^L e o resultado da computação de M^L sobre x por $M^L(x)$.

Se uma linguagem L' é decidida por uma máquina de Turing com acesso a um oráculo O nós dizemos que L é decidível em relação a O.

A seguir nós vemos que, como esperado, a adição de um oráculo nos dar um poder adicional em relação a máquinas de Turing convencionais.

Teorema 2.5. *Existe uma linguagem que é decidível em relação a algum oráculo mas que não é decidível por uma máquina de Turing sem acesso a nenhum oráculo.*

Demonstração. Considere a seguinte linguagem:

$$\mathsf{HALT} = \{\langle \alpha, x\rangle |\ M_\alpha \text{ para após um número finito de passos quando recebe x como entrada}\}$$

Podemos decidir HALT com um oráculo para HALT. M^{HALT} simplesmente copia o conteúdo de sua fita de entrada para a sua fita de oráculo e faz uma consulta ao oráculo. Após isso M^{HALT} escreve em sua fita de saída 1 se ela estiver no estado q_{sim}, ou 0 caso esteja no estado $q_{\text{não}}$.

Pelo teorema seguinte nós vemos que nenhuma máquina de Turing convencional decide HALT.

□

Teorema 2.6 ([Tur36])**.** HALT *não é decidida por nenhuma máquina de Turing sem acesso a um oráculo.*

Demonstração. Assuma que H decida HALT. Neste caso é possível simular a execução de H sobre qualquer entrada em tempo finito. Seja H' uma máquina de Turing tal que

$$H' \text{ rejeita } x \iff M_x \text{ aceita a entrada } x$$

H' primeiro simula H sobre a entrada $(\langle M_x \rangle, x)$ e aceita após este passo se e somente se a simulação rejeita. Após isso, H' simula M_x sobre a entrada x e aceita se e somente a simulação rejeita. Se denotarmos por $\mathbb{1}_M$ a função característica da máquina de Turing M, ou seja,

$$\mathbb{1}_M(x) = \begin{cases} 1 & \text{se } M \text{ aceita a entrada } x \\ 0 & \text{caso contrário} \end{cases}$$

temos que a função característica de H' pode ser escrita como

$$\mathbb{1}_{H'}(x) = 1 - \mathbb{1}_H(\langle M_x \rangle, x)\mathbb{1}_{M_x}(x).$$

Pela nossa hipótese, todos os passos que H' faz pode ser feito em tempo finito e portanto, H aceita a entrada $(\langle H' \rangle, x)$, para todas as strings $x \in \{0,1\}^*$. Em particular,

$$H(\langle H' \rangle, \langle H' \rangle) = 1 \Rightarrow \mathbb{1}_H(\langle H' \rangle, \langle H' \rangle) = 1.$$

Daí temos que

$$\begin{aligned} \mathbb{1}_{H'}(\langle H' \rangle) &= 1 - \mathbb{1}_H(\langle H' \rangle, \langle H' \rangle)\mathbb{1}_{H'}(\langle H' \rangle) \\ &= 1 - \mathbb{1}_{H'}(\langle H' \rangle), \end{aligned}$$

uma contradição.

□

A técnica de prova usada acima se chama *diagonalização*. Esta técnica foi inventada por Georg Cantor que a usou para provar que existe diferente níveis de infinito. Mais precisamente, ele provou que a cardinalidade do conjuntos de todas as string binárias de tamanho infinito tem cardinalidade maior do que o conjunto de todos os números naturais, apesar de ambos os conjuntos serem infinitos.

2.4 Circuitos booleanos

Agora nós vamos ver circuitos booleanos que é o principal modelo de computação para o propósito deste trabalho. Nós também iremos ver como circuito booleanos estão naturalmente relacionados com fórmulas booleanas. Ambos os modelos são "flexíveis" no sentido em que eles não estão somente restritos a um conjunto fixo de operações permitidas. Também iremos ver que os dois modelos são equivalentes dado que as operações primitivas permitidas são as mesmas.

Um circuito booleano é um grafo direcionado acíclico. Nós particionamos os vértices do circuito em três partes: 1) n entradas do cirtuito 2) k portas lógicas 3) uma porta de saída. As entradas do circuito têm grau de entrada zero e os vértice de saída têm grau de saída também zero.

Uma base Ω é uma coleção finita e não vazia de funções booleanas. Cada porta lógica de um circuito (incluindo a porta de saída) deve computar uma função booleana tirada de uma base Ω. Os vértices de entrada guardam algum valor booleano (0 ou 1).

O valor da computação de um circuito vai depender dos valores das variáveis de entrada e de uma sequência de valores de funções tirada de Ω que dependem das variáves de entrada e/ou de funções previamente computadas.

Definição 2.7. *(Circuitos booleanos)*

Um circuito booleano C sobre uma base Ω é um grafo direcionado acíclico com m vértices donde n vértices de grau de entrada zero são as variáveis de entrada $v_1, \ldots, v_n$, e todos os outros vértices $v_{n+1}, \ldots, v_m$ são portas lógicas que computam alguma função em Ω e que têm grau de entrada e grau de saída maior ou igual a um com a exceção de v_m que é a saída do circuito e tem grau de saída zero.

Cada vértice v do circuito terá um valor associado a ele que denotamos por $val(v)$. As arestas que chegam em uma porta lógica são suas entradas enquanto que as arestas que saem dela são as suas saídas. Dessa forma, se dois vértices u e v, onde u é uma porta lógica ou uma variável de entrada e v é uma porta lógica, são ligados por uma aresta que sai de u e chega em v, então temos que $val(v)$ depende de $val(u)$. Mais precisamente, se v é uma porta lógica que computa uma função $g \in \Omega$ e $u_1, \ldots, u_l$ são todos os vértices que são predecessores de v, então o valor de v é definido por $val(v) = g(val(u_1), \ldots, val(u_l))$.

Como um circuito é um grafo direcionado acíclico e os n primeiros vértices v_1 até v_n são fontes e o último vértice v_m é um sumidouro, podemos assumir que o ordenamento $(v_1, v_2, \ldots, v_m)$ é um ordenamento topológico dos vértices do circuito. Portanto podemos formalizar o funcionamento do circuito da seguinte maneira: Assume-se que os vértices de entradas $v_1, \ldots, v_n$ recebem valores booleanos arbitrários e $x = val(v_1) \cdot val(v_2) \ldots val(v_{n-1}) \cdot val(v_n)$ é a entrada do circuito e queremos computar o valor $C(x) = val(v_m)$. Para isso segue-se em m - n passos onde no i-ésimo passo é computado $val(v_{n+i})$. Note que em cada passo os valores dos vértices dos quais v_{n+i} depende já foram decididos por causa da nossa hipótese que $(v_1, \ldots, v_m)$ é um ordenamento topológico dos vértices do circuito.

Se $f : \{0,1\}^n \to \{0,1\}$ é a função booleana $f(x) = C(x)$, para todo $x \in \{0,1\}^n$, então é dito que C computa f.

O fan-in de uma porta lógica é o número de entradas que ela aceita e o fan-out é o número de saídas (ou seja, o grau de saída). Geralmente o fan-in das porta lógicas de um circuito vão ser limitados por uma constante mas em alguns casos nós vamos considerar classes de circuitos onde não há nenhuma restrição quanto ao fan-in máximo das portas lógicas.

Para superar a limitação de circuitos aceitarem somente entradas de um tamanho fixo nós definimos uma sequência infinita de circuitos onde o n-ésimo circuito da sequência computa uma função com entradas de tamanho n. Desta forma podemos falar de circuitos (ou família de circuitos) que decidem uma dada linguagem, ao invés de somente computar uma função com domínio nas string binárias de um determinado tamanho.

Definição 2.8. *(Família de circuitos)*

Uma família de circuitos é uma sequência $\{C_n\}_{n\in\mathbb{N}}$ de circuitos booleanos onde cada C_n computa uma função $f_n : \{0,1\}^n \to \{0,1\}$.

Dizemos que $\{C_n\}_{n\in\mathbb{N}}$ computa uma função $f : \{0,1\}^* \to \{0,1\}$ se $f(x) = C_{|x|}(x)$, $\forall x \in \{0,1\}^*$.

Adicionalmente, se uma família de circuitos $\{C_n\}_{n\in\mathbb{N}}$ computa a função característica de uma linguagem L qualquer, então dizemos que $\{C_n\}_{n\in\mathbb{N}}$ decide L.

2.4.1 Fórmulas booleanas

Uma *fórmula booleana* é um circuito onde todas as porta lógicas têm fan-out igual a 1. Todos circuitos booleanos podem ser convertidos para uma fórmula se substituirmos todas portas lógicas com fan-out maior do que um por um número suficiente de cópias dessas portas lógica com somente uma saída. E como fórmulas são um caso especial de circuitos temos que os dois modelos são equivalentes. Geralmente descrevemos fórmulas lógicas através de variáves, conectivos e parênteses para denotar a sequência correta de operações. Por exemplo, considere os seguintes operadores lógicos:

a	b	$a \vee b$
0	0	0
0	1	1
1	0	1
1	1	1

a	b	$a \wedge b$
0	0	0
0	1	0
1	0	0
1	1	1

a	$\neg a$
0	1
1	0

O operador $\vee$ é chamado de *OU*, $\wedge$ é chamado de *E* e $\neg$ de NÃO. A base formada por $\vee, \wedge$ e $\neg$ é a mais "popular" no contexto de operação lógicas. Se x_1 e x_2 são variáveis então $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$ é um exemplo de fórmula lógica.

Alternativamente, podemos definir fórmulas logicas sobre a base $\{\vee, \wedge, \neg\}$ recursivamente da seguinte forma:

- Se x é uma variável então x é uma fórmula
- Se ϕ e ψ são fórmulas então também são $\phi \vee \psi$, $\phi \wedge \psi$ e $\neg\phi$.

A seguir nós vemos algumas forma normais de se representar fórmulas lógicas que vão ser bastante úteis para nós.

Definição 2.9. *(Forma normal conjuntiva (FNC))*

Uma fórmula lógica sobre as variáveis $x_1, \ldots, x_n$ é dita estar na forma normal conjuntiva (ou FNC) se ela é o E de OUs de variáveis em $\{x_1, \ldots, x_n\}$ ou as suas negações.

Ou seja, uma fórmula na FNC pode ser escrita como

$$\bigwedge_{i=1}^{m} c_i$$

Onde os $c_i = x_{i_1} \vee \cdots \vee x_{i_{k(i)}}$ são chamados de cláusulas e m é o número de claúsulas na fórmula.

Por exemplo, se ϕ é uma fórmula sobre as variáves x_1, x_2, x_3 e x_4, então

$$\phi = (x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_3) \wedge (\overline{x_2} \vee x_4)$$

está na forma normal conjuntiva.

Uma fórmula é dita ser uma k-FNC se ela estar na forma normal conjuntiva e cada cláusula estiver restrita a no máximo k literais.

Definição 2.10. *(Forma normal disjuntiva)*

Uma fórmula lógica sobre as variáveis $x_1, \ldots, x_n$ é dita estar na forma normal disjuntiva (ou FND) se ela é o OU de Es de variáveis em $\{x_1, \ldots, x_n\}$ ou as suas negações.

Os Es são chamados de termos. Se o número de termos na fórmula for m e $c_i = x_{i_1} \wedge \cdots \wedge x_{i_{k(i)}}$, $i \in [m]$, forem termos então uma fórmula na FND pode ser escrita como

$$\bigvee_{i=1}^{m} c_i$$

Por exemplo, a fórmula $\phi = (x_1 \wedge \overline{x_2}) \vee (x_1 \wedge x_3) \vee (\overline{x_2} \wedge x_4)$ está na forma normal disjuntiva.

Assim como k-FNCs, uma k-FND é uma fórmula na forma normal disjuntiva com a restrição que cada termo deva ter no máximo k literais.

Definição 2.11. *Um* mintermo *é o E de todas as variáves de uma fórmula ou suas negações. Por exemplo,* $x_1 \wedge \neg x_2 \wedge x_3 \wedge \neg x_4$ *é um mintermo sobre as variáveis* x_1, x_2, x_3 *e* x_4*.*

Considere a seguinte operação sobre uma variável booleana x:

$$x^b = \begin{cases} x & \text{caso } b = 1, \\ \overline{x} & \text{caso } b = 0. \end{cases}$$

Então podemos escrever um mintermo sobre as variáveis $x_1, \ldots, x_n$ de uma fórmula booleana como $x_i^{b_1} \wedge \cdots \wedge x_n^{b_n}$, onde cada b_i é 0 ou 1. Daí fica óbvio que um mintermo é verdadeiro se e somente se $x_i = b_i$, para cada $i \in [n]$. Se $x = (b_1, \ldots, b_n)$ é uma atribuição às variáveis então associamos o seguinte mintermo a esta atribuição:

$$\bigwedge_{i=1}^{n} x_i^{b_i}$$

Então, para cada função booleana f com n variáveis, o n-FND onde cada termo é um mintermo associado às atribuições que satisfazem $f(x) = 1$ é uma fórmula que computa f, consequentemente todas funções booleanas podem ser computadas por um circuito booleano e todas linguagens em $\{0,1\}^*$ são decididas por alguma família de circuitos, incluindo a linguagem HALT que não é computável por máquinas de Turing convencionais.

Algumas vezes nós vamos chamar mintermos de monômios e podemos denotar eles por algo como $x_1^{\alpha_1} \ldots x_n^{\alpha_n}$, onde $\alpha_i \in \{0,1\}$.

Árvores de decisão

Árvores de decisão são uma outra forma de representar funções Booleanas. Uma árvore de decisão que computa uma função $f : \{0,1\}^n \to \{0,1\}$ pode ser vista como um algoritmo de consulta T que funciona da seguinte forma.

1. Existe uma consulta inicial q_1 à uma variável x_{i_1} para algum $i_1 \in [n]$ e duas consultas $T(q_1, 0)$ e $T(q_1, 1)$ em que a próxima consulta é $T(q_1, b)$ se for o caso que a consulta à variável x_{i_1} retorna o bit b (ou seja $x_{i_1} = b$).
2. Para todas as outras consultas q_j subsequentes, nós definimos $T(q_j, b)$ de forma que a próxima consulta é $T(q_j, b)$ se a consulta à variável x_{i_j} retorna b.
3. Eventualmente, após T ter feito um número suficiente de consultas às variáveis ele dá como saída o valor $f(x) = b$ de forma que toda as strings $x' \in \{0,1\}^n$ que são consistentes com as respostas às consultas feitas por T satisfazem $f(x') = b$.

Uma sequência $(q_1, r_1), (q_2, r_2), \ldots, (q_l, r_l)$ de consultas e respostas forma um caminho de tamanho l. Uma árvore de decisão é definida da seguinte forma.

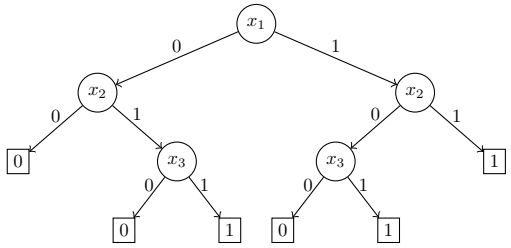
Definição 2.12 (Árvores de decisão)**.** *Uma árvore de decisão que computa a função* $f : \{0,1\}^n \to \{0,1\}$ *é um algoritmo de consulta em que a primeira consulta é a raíz da árvore e todo caminho acaba em um nodo chamado de folha. Todos os outros nodos da árvore são chamados de nodos internos. Cada nodo que não é uma folha guarda uma variável* x_i*, para* $i \in [n]$ *e cada folha guarda um bit* $b \in \{0,1\}$*. Nós exigimos que*

1. *Cada nodo que não é uma folha tenha exatamente dois descendentes e cada aresta que liga este nodo a seus dois descendentes devem estar marcado com 0 ou 1.*

2. *Se um nodo interno estiver marcado com a variável x_j então o caminho pela árvore segue a aresta marcada com o bit b em que $x_j = b$.*

3. *Nenhuma variável é consultada mais do que uma vez em qualquer caminho da árvore.*

4. *Se $x \in \{0,1\}^n$ segue um caminho que acaba em uma folha que guarda o bit b então $f(x) = b$.*



A figura (...) dá um exemplo de uma árvore de decisão. Pode-se verifica que esta árvore de decisão computa a função Majority_3 em que $\text{Majority}_3(x)$ é 1 se e somente se o número de 1s em x é maior do que 1.

A profundidade de uma árvore de decisão é o tamanho do maior caminho da raiz até uma folha. O tamanho de uma árvore de decisão é o número de folhas que ela tem.

Nós podemos ver que toda árvore de decisão pode ser convertida em uma fórmula FNC ou FND.

2.5 Complexidade computacional

Na seção anterior nós vimos que Turing nos deu uma definição formal do que nós intuitivamente pensamos ser computável. Porém, no mundo real, um problema ter um processo computacional finito que o resolva não é suficiente. Também queremos que a computação seja feita num tempo que seja útil para nós. O que foi observado é que o tempo de execução de algoritmos em computadores cresce a medida em que o tamanho da entrada também cresce. Pense no tamanho da entrada sendo medido como, por exemplo, o número de bits na representação binária de um número ou o número de vértices em um grafo. Como normalmente é desejável que um algoritmo seja eficiente para entradas de tamanho razoavelmente grande (em alguns casos o tamanho da entrada pode ser 10^6, por exemplo), nós queremos que a função de crescimento do algoritmo não cresça muito rápidamente — para que até para entradas de tamanho "razoavelmente grande" o número de passos necessários para realizar o algoritmo não seja excessivamente grande. Portanto foi importante definir uma forma de medir a complexidade de algoritmos e também o que nos queremos dizer por uma "função eficiente"para o tempo de execução de um algoritmo.

Talvez o primeiro artigo que definiu uma medida de complexidade para problema computacionais foi [HS65], onde Hartmarnis e Stearns definiram que uma sequência binária α é computável em tempo T, onde T é uma função computável monotônica crescente de $\mathbb{N}$ para $\mathbb{N}$, se existe uma máquina de Turing que dá como saída o n-ésimo bit de α em menos do que $T(n)$ passos. Em [Edm65], Edmonds propõe que devemos considerar funções polinomiais como sendo sinônimo de eficiência.

Nós dizemos que uma máquina de Turing M roda em tempo $T(n)$ se M, ao receber uma entrada de tamanho n, executa no máximo $T(n)$ passos (um passo da computação da máquina de Turing envolve escrever símbolos em suas fitas de trabalho, movimentar os cabeçotes de suas fitas e mudar o seu estado atual). Ao longo deste trabalho nós vamos assumir que esta função T e qualquer outra função que estivermos usando para medir a complexidade de um problema computacional é *tempo-construtível* da forma definida a seguir.

Definição 2.13 (Funções tempo-construtíveis)**.** *Uma função $f : \mathbb{N} \to \mathbb{N}$ é dita ser tempo-construtível se existe uma máquina de Turing M_f que ao receber a string unária 1^n em sua entrada, M_f escreve a representação binária de $f(n)$ em sua fita de saída em $\mathcal{O}(f(n))$ passos.*

Todas funções que nos interessam como as funções polinomiais, $\log n$, 2^{n^c}, etc. são tempo construtíveis.

Nós vimos no teorema 2.2 que existe uma máquina de Turing que pode simular a execução de todas as outras máquinas de Turing sobre qualquer entrada. Também foi dito que a simulação é "eficiente" e que segundo Edmonds isto deveria significar que a simulação pode ser feito em tempo polinomial. E nós podemos verificar que o número de passos que $\mathcal{U}$ precisa para simular uma máquina de Turing M de tempo $T(n)$ é $\mathcal{O}(T(n)^2)$. Para provar isto temos que ver quanto passos $\mathcal{U}$ necessita para simular um único passo de M. Em cada simulação de um passo, $\mathcal{U}$ visita cada "espaço" que representa uma fita de M, e como uma computação que executa menos do que $T(n)$ passos não pode usar mais do que $T(n)$ células de sua fita, temos que cada espaço contém no máximo $T(n)$ células. Então para simular um passo de M, $\mathcal{U}$ visita algo em torno de $kT(n)$ células de sua fita de trabalho, onde k é o número de fitas de M, e portanto o "slowdown" de simular M é apenas $\mathcal{O}(T(n))$.

Nós podemos fazer melhor do que $T(n)^2$, nós podemos simular uma máquina de Turing com "slowdown" logarítmico.

Teorema 2.14. *Existe uma máquina de Turing $\mathcal{U}^*$ que sobre a entrada (α, x), $\mathcal{U}^*$ dá como saída $M_\alpha(x)$. Além disso, se $T(|x|)$ é o tempo que M_α leva para executar sua computação sobre a entrada x, então $\mathcal{U}^*$ roda em tempo $\mathcal{O}(T(|x|) \log T(|x|))$ ao receber (α, x) em sua fita de entrada.*

Uma prova do teorema 2.14 pode ser encontrada em [AB09] (Teorema 1.9). O resultado foi originalmente proposto por Hennie e Stearns [HS66].

Construindo sobre o resultado acima nós podemos provar o seguinte resultado que nos será útil mais para frente.

Definição 2.15. *Uma máquina de Turing* oblivious *é uma máquina de Turing cuja o movimento de seus cabeçotes de fita só dependem do tamanho da entrada, e não no conteúdo das células e o estado atual.*

Desta forma, a função de transição de uma máquina de Turing *oblivious* $A = \{\Gamma, Q, \delta\}$ de k fitas é $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1}$. Nós podemos imaginar que existe uma função $m : \mathbb{N} \to \{\{E, N, D\}^k\}^*$ tal que ao receber uma entrada x, os movimentos dos cabeçotes das fitas de A é dado por $m(|x|) = (z_1, \ldots, z_{T(|x|)})$, onde cada $z_i \in \{E, N, D\}^k$ representa os movimentos dos cabeçotes de fita de A no i-ésimo passo e $T(|x|)$ é o tempo em que A para ao receber entradas de tamanho $|x|$.

Teorema 2.16. *Se M é uma máquina de Turing que roda em tempo $T(n)$ para entradas de tamanho n, então existe uma máquina de Turing* oblivious *A de duas fitas que roda em tempo $\mathcal{O}(T(n) \log T(n))$ tal que $A(x) = M(x)$, para todo $x \in \{0, 1\}^*$.*

Demonstração. Nós podemos modificar a MT $\mathcal{U}^*$ no teorema 2.14 de forma que ela seja uma máquina de Turing *oblivious* sem aumentar significativamente o seu tempo de execução. E além disso, $\mathcal{U}^*$ pode ser construida usando somente 2 fitas.

Então, A simplesmente executa a simulação de $\mathcal{U}^*$ sobre entradas $(\langle M \rangle, x)$, para qualquer $x \in \{0, 1\}^*$.

□

Uma das contribuições de Hartmanis e Stearns em [HS65] foi que eles mostraram como podemos agrupar problemas computacionais de acordo com o número de passos que um máquina de Turing necessita para resolvê-los. Nesta seção iremos nos preocupar apenas com o tempo e espaço necessários para resolver problemas

computacionais. Algumas classes de complexidade de tempo são definidas a seguir, e no fim desta seção iremos ver algumas classes de complexidade de espaço.

Definição 2.17. *Para uma função $T : \mathbb{N} \to \mathbb{N}$, nós definimos as seguintes classes de problemas:*

- $\mathsf{DTIME}(T(n))$*: a classe de todas linguagens L tal que existe uma máquina de Turing determinística M de tempo $T(n)$ que decide L.*
- $\mathsf{NTIME}(T(n))$*: a classe de todas linguagens L tal que existe uma máquina de Turing não-determinística N que decide L e N executa no máximo $T(n)$ passos ao receber uma entrada de tamanho n, indepedente das escolhas das funções de transição de N.*
- $\mathsf{coDTIME}(T(n))$*: a classe de todas linguagens L tal que $\overline{L} \in \mathsf{DTIME}(T(n))$, onde $\overline{L}$ é o complemento da linguagem L (ou seja, $x \in L \iff x \notin \overline{L}$). Da mesma forma nós definimos* $\mathsf{coNTIME}(T(n))$.

As classes P e NP

Como já vimos, iremos usar tempo polinomial como sinônimo de eficiência. Uma linguagem L é decidida em tempo polinomial se existe um polinômio p tal que o tempo necessário para decidir a pertinência de uma string x em L é menor do que $p(|x|)$. A classe de linguagens decididas em tempo polinomial é chamada de P.

Definição 2.18 (A classe P)**.** *Uma linguagem L é dita estar em* P *se e somente se existe $c \geq 1$ tal que $L \in \mathsf{DTIME}(n^c)$.*

Um dos grandes objetivos de designers de algoritmos é provar que um determinado problema está em P pois então geralmente ele pode ser implementado eficientemente em um computador. Alguém poderia dizer que talvez exista um problema (natural) que esteja em P mas o tempo de execução do algoritmo para este problema é algo do tipo $10^{1000}n$ ou n^{1000}, o que com certeza não seria eficiente nem mesmo para $n = 2$. É verdade que um problema estar em P não implica necessariamente em ele poder ser resolvido eficientemente. Na verdade, nem mesmo a não existência de um algoritmo de tempo polinomial para um problema implica em ele não poder ser resolvido eficientemente na prática. Mas usar a convenção de tempo polinomial = eficiência é conveniente quando estamos estudando classes de complexidade e a relação entre elas, por alguns motivos como por exemplo algumas modificações na definição de máquinas de Turing e até mesmo outros modelos computacionais mais realistas (como máquinas de acesso aleatório) não alteram a classe P, entre outros motivos.

Enquanto que P procura capturar linguagens que podem ser decididas eficientemente, a classe NP por sua vez procura capturar linguagens cuja suas instâncias sejam eficientemente verificáveis.

Definição 2.19 (A classe NP)**.** *Uma linguagem L está em* NP *se e somente se existe um polinômio p e uma máquina de Turing de tempo polinomial M tal que $x \in L \iff \exists u \in \{0,1\}^{p(|x|)} M(x,u) = 1$.*

Dizemos que u é um certificado da pertinência de x em L. Nós podemos definir NP de uma outra forma:

Definição 2.20. $\mathsf{NP} = \bigcup_{c \geq 1} \mathsf{NTIME}(n^c)$.

Para ver que as duas definições são equivalentes note que as escolhas da máquina de Turing não-determinística podem servir como um certificado, enquanto que uma máquina de Turing não-determinística poderia "adivinhar" um certificado para x.

A questão em aberto mais importante em complexidade computacional pergunta se as classes P e NP são iguais. Esse problema tem alguma importância histórica já que vários problemas que são importante em aplicações práticas que estão em NP não parecem, pelo que sabemos até agora, ter solução melhor do que tentar exaustivamente todas as possibilidades, a busca por uma solução para esses problemas melhor do que a busca exaustiva esteve no coração de algumas das primeiras pesquisas em complexidade computacional.

Definição 2.21 (A classe coNP)**.** $\mathsf{coNP} = \bigcup_{c \geq 1} \mathsf{coNTIME}(n^c)$.

Note que $\mathsf{P} = \mathsf{coP}$, já que um procedimento que decide eficientemente a pertinência de uma *string* em uma linguagem também pode ser usada para decidir a não pertinência (simplesmente inverta a saída), e portanto $\mathsf{P} = \mathsf{NP}$ implica em $\mathsf{NP} = \mathsf{coNP}$.

Reduções

Um dos principais conceitos em teoria da computação é o de uma *redução*. Uma redução é basicamente um procedimento que transforma uma instância de um problema A em uma instância de um outro problema B.

Definição 2.22 (Reduções)**.** *Uma redução de um problema $L \subseteq \{0,1\}^*$ para um problema $L' \subseteq \{0,1\}^*$ é uma função $f : \{0,1\}^* \to \{0,1\}^*$ tal que $x \in L \iff f(x) \in L'$, para todo $x \in \{0,1\}^*$.*

Além disso, L é dita ser redutível em tempo polinomial para L', o que denotamos por $L \leq_p L'$, se a redução f pode ser computada em tempo polinomial.

Reduções de tempo polinomial vão ser útil quando formos ver o próximo assunto. Se L é redutível em tempo polinomial para L', então um algoritmo de tempo polinomial para L' implica em um algoritmo de tempo polinomial para L, já que podemos usar a redução f para mapear uma string $x \in \{0,1\}^*$ em uma instâcia $f(x)$ de L' e depois usamos o algoritmo A de tempo polinomial que decida L' para computar $A(f(x))$. Se o tempo necessário para computar A sobre entradas de tamanho n for $p(n)$, onde p é um polinômio, e q for o tempo necessário para computar f então acabamos de mostrar que podemos decidir L em tempo menor do que $q(|x|) + p(q(|x|))$.

NP-completude e o teorema de Cook-Levin

Algumas linguagens em uma determinada classe de complexidade tem uma propriedade interessante em que elas capturam toda a dificuldade daquela classe. Um linguagem L é completa para uma classe sobre uma determinada "classe de reduções" $\leq_R$ (por exemplo, reduções em tempo polinomial como vimos na definição 2.22) se ela pertence à classe e todos os outros problemas dentro desta classe são redutíveis através de $\leq_R$ para L.

Definição 2.23 (NP-completude)**.** *Uma linguagem L é dita ser NP-difícil sse para todas linguagens $A \in \mathsf{NP}$, $A \leq_p L$.*

Se além de ser NP-difícil L também está em NP então dizemos que L é NP-completa.

Problemas NP-completos (que sejam naturais) existem, como foi provado por Stephen Cook e Leonid Levin, independentemente, no começo da década de 70. [Coo71, Lev73] O primeiro problema que foi provado ser NP-completo foi o problema da satisfazibilidade booleana.

Definição 2.24 (O problema da satisfazibilidade booleana)**.** *No problema da satisfazibilidade booleana, que chamaremos de SAT, é dado uma fórmula ϕ com variáveis $x_1, \ldots, x_n$ e queremos de decidir se existe uma atribuição $(x'_1, \ldots, x'_n)$ às variáves $x_1, \ldots, x_n$ tal que $\phi(x'_1, \ldots, x'_n) = 1$.*

Teorema 2.25 (Teorema de Cook-Levin)**.** SAT *é NP-completo.*

Apesar de poder ser um pouco longa, a prova do teorema 2.25 é bem simples de entender. Basicamente, nós temos que se A é uma linguagem em NP, então existe uma máquina de Turing M (que podemos assumir ter apenas uma fita) que aceita uma entrada $x \in \{0,1\}^n$ com um certificado $u \in \{0,1\}^{\mathrm{poly}(n)}$ se e somente se $x \in A$ e u é um certificado da pertinência de x em A. A função $f : \{0,1\}^* \to \{0,1\}^*$ que transforma x em uma fórmula ϕ_x que é satisfazível se e somente se $x \in A$ faz o seguinte:

1. Se para todo $n > 0$ M roda em tempo menor do que $T(n)$ sobre entradas de tamanho n então f constroi um tableau $T(|x|) \times T(|x|)$ onde a i-ésima linha deste tableau guardará a configuração de M no seu i-ésimo passo.

2. A fórmula ϕ_x tem $T(|x|)^2$ variáveis que chamaremos de v_{ij}, $1 \leq i, j \leq T(|x|)$. O valor da variável v_{ij} é o conteúdo da célula na linha i e coluna j do tableau.

3. Pela computação de uma máquina de Turing ser local, o que significa dizer que o conteúdo de uma das células em um passo da computação depende somente do estado atual, da posição do cabeçote da fita e do conteúdo das duas células adjacente à ela, podemos construir uma fórmula booleana que decide o valor da variável v_{ij} em função das variáveis $v_{(i-1)(j-1)}, v_{(i-1)j}$ e $v_{(i-1)(j+1)}$. Esta fórmula depende somente da função de transição de M e portanto tem tamanho constante.

4. Precisamos assegurar algumas outras coisas, como por exemplo que a primeira linha do tableau é uma configuração inicial válida e que a última linha é uma configuração de aceitação (isto é, uma configuração onde o estado atual é q_{aceita}).

Pela natureza "repetitiva" da redução e pela fórmula ter tamanho polinomial ($\mathcal{O}(T(n)^2)$) podemos ver que ela pode ser feita em tempo polinomial. Finalmente, SAT $\in$ NP já que uma atribuição das variáveis que satisfazem uma fórmula pode servir como certificado.

Agora que nós temos um único problema que sabemos ser NP-completo, nós podemos provar que outros problemas são também NP-completo mostrando que SAT é redutível em tempo polinomial para eles. Isso segue pois a relação $\leq_p$ é transitiva. Por exemplo, podemos provar que a linguagem 3-SAT, que pergunta se uma fórmula na 3-FNC é satisfazível, é NP-completa. Em 1972, Richard Karp publicou [Kar72] onde 21 problemas importantes foram provados serem NP-completos e deste então milhares de problemas que aparecem em aplicações práticas já foram provados serem NP-completos. O livro de Garey e Johnson é uma excelente referência para o fenômeno da NP-completude. [GJ02]

Como já observamos antes, se existe uma linguagem NP-completa em P então P = NP, pois poderiamos usar a redução de tempo polinomial para L e depois o seu algoritmo de tempo polinomial para decidir qualquer outra linguagem em NP em tempo polinomial.

Hierarquia polinomial

Considere o seguinte problema em NP:

CLIQUE. *Dado um inteiro $k > 0$ e um grafo G, aceite se G tem um clique de tamanho maior ou igual a k.*

Podemos ver que CLIQUE está em NP pois um clique de tamanho maior ou igual a k em G é obviamente um certificado que G tem um clique de tamanho maior ou igual a k. Mas se ao invés de decidir se G tem um clique de tamanho pelo menos k nós queremos decidir se o maior clique em G tem tamanho k, como no problema MAX-CLIQUE:

MAX-CLIQUE. *Dado um inteiro $k > 0$ e um grafo G, aceite se o maior clique em G tem tamanho igual a k.*

Agora não fica tão óbvio para nós o que seria um certificado para MAX-CLIQUE. Além de termos que mostrar um clique de tamanho k, também devemos mostrar que nenhum subconjunto de tamanho maior do que k dos vértices de G formam um clique. Porém, adicionando um quantificador $\forall$ parece ser o suficiente para nós podermos capturar problemas como MAX-CLIQUE, o que a classe NP não parece conseguir fazer pelo o que nós sabemos até agora.

Definição 2.26 (A classe Σ_2^p)**.** *Uma linguagem L é dita estar em Σ_2^p se e somente se existe um polinômio p e uma máquina de Turing M que roda em tempo $p(n)$ tal que L pode ser escrita como:*

$$\text{Para todo } x \in \{0,1\}^*,\ x \in L \iff \exists x_1 \in \{0,1\}^{p(|x|)} \forall x_2 \in \{0,1\}^{p(|x|)} M(x, x_1, x_2) = 1$$

Se as instâncias MAX-CLIQUE são da forma $\langle G, k\rangle$, onde G é um grafo e $k > 0$ um inteiro, então dizer $\langle G, k\rangle \in$ MAX-CLIQUE é o mesmo que dizer que *existe* um clique de tamanho k e que *todos* subconjuntos de tamanho maior do que k dos vértices de G não formam um clique.

Nós podemos ainda generalizar as classes NP e Σ_2^p, o que nós chamamos de hierarquia polinomial:

Definição 2.27 (Hierarquia polinomial)**.** *Para $k \geq 1$, uma linguagem L é dita estar em Σ_k^p se L pode ser expressa da seguinte forma:*

$$x \in L \iff \exists x_1 \in \{0,1\}^{p(|x|)} \forall x_2 \in \{0,1\}^{p(|x|)} \dots Q_k x_k \in \{0,1\}^{p(|x|)} M(x, x_1, x_2, \dots, x_k) = 1$$

Onde Q_k é $\exists$ se k é ímpar ou $\forall$ se k é par. M é uma máquina de Turing de tempo $p(n)$.

A hierarquia polinomial é $\mathsf{PH} = \bigcup_{k\geq 1} \Sigma_k^p$.

Note que $\mathsf{NP} = \Sigma_1^p$ e também podemos chamar P de Σ_0^p.

Assim como fizemos com NP, também podemos generilizar a classe coNP através de quantificadores alternantes. A diferença é que o primeiro quantificador é um $\forall$.

Definição 2.28. *Para todo $k \geq 1$ a classe $\mathsf{co}\Sigma_k^p$ consiste de todas as linguagens L que podem ser expressas como:*

$$x \in L \iff \forall x_1 \in \{0,1\}^{p(|x|)} \exists x_2 \in \{0,1\}^{p(|x|)} \dots Q_k x_k \in \{0,1\}^{p(|x|)} M(x, x_1, x_2, \dots, x_k)$$

Onde agora Q_k é $\forall$ se k é ímpar e $\exists$ caso contrário. E de novo, M é uma máquina de Turing de tempo $p(n)$. Para cada k, nós chamamos $\mathsf{co}\Sigma_k^p$ de Π_k^p.

E nós temos que $\mathsf{coNP} = \Pi_1^p$.

É fácil ver que para todo $k \geq 1$ nós temos as seguintes desigualdades:

$$\Sigma_k^p \subseteq \Pi_{k+1}^p \subseteq \Sigma_{k+2}^p$$

Portanto $\mathsf{PH} = \bigcup_{k\geq 1} \Pi_k^p$

Nós dizemos que a hierarquia polinomial *colapsa* se para algum k, $\mathsf{PH} = \Sigma_k^p$. Neste caso dizemos que a hierarquia polinomial colapsa para o seu k-ésimo level e também temos que $\Sigma_k^p = \Sigma_l^p$, para todo $l > k$.

Teorema 2.29. *Para todo $k \geq 1$, se $\Sigma_k^p = \Pi_k^p$ então $\mathsf{PH} = \Sigma_k^p$.*

Demonstração.

Assuma que para algum $k \geq 1$, $\Sigma_k^p = \Pi_k^p$.

Nós mostramos por indução que para todo $l > k$, $\Sigma_l^p = \Pi_l^p = \Sigma_k^p$. Para isso só precisamos mostrar que $\Sigma_l^p \subseteq \Sigma_k^p$, pois por nós termos assumido que $\Sigma_k^p = \Pi_k^p$, $\Sigma_l^p = \Sigma_k^p$ implica em Σ_l^p estar fechada sob complemento.

Para $l = k + 1$, nós temos que toda linguagem L em Σ_{k+1}^p pode ser expressa como:

$$x \in L \iff \exists x_1 \in \{0,1\}^{p(|x|)} \forall x_2 \in \{0,1\}^{p(|x|)} \dots Q_{k+1} x_{k+1} \{0,1\}^{p(|x|)} M(x, x_1, x_2, \dots, x_{k+1}) = 1 \tag{2.7}$$

para alguma polinômio p e máquina de Turing de tempo polinomial M.

Então considere a seguinte linguagem L':

$$(x, x_1) \in L' \iff \forall x_2 \in \{0,1\}^{p(|x|)} \dots Q_{k+1} x_{k+1} \{0,1\}^{p(|x|)} M(x, x_1, x_2, \dots, x_{k+1}) = 1$$

L' está em $\Pi_k^p = \Sigma_k^p$ portanto podemos reescrever L' como:

$$(x, x_1) \in L' \iff \exists y_1 \in \{0,1\}^{p(|x|)} \forall y_2 \in \{0,1\}^{p(|x|)} \dots Q_k x_k \in \{0,1\}^{p(|x|)} M'(x, x_1, y_1, y_2, \dots, y_k) = 1 \tag{2.8}$$

Então podemos trocar toda parte a partir do primeiro quantificador $\forall$ de 2.7 pelo lado direito de 2.8 e temos o seguinte:

$$x \in L \iff \exists x_1, y_1 \in \{0,1\}^{p(|x|)} \forall y_2 \in \{0,1\}^{p(|x|)} \dots Q_k y_k \in \{0,1\}^{p(|x|)} M'(x, x_1, y_1, y_2, \dots, y_k) = 1$$

Portanto $L \in \Sigma_k^p$.

Para provar para outros valores de $l > k+1$, nós provamos da mesma maneira mas assumindo que $\Sigma_{l-1}^p = \Pi_{l-1}^p$ que agora sabemos que são iguais a Σ_k^p.

□

Assim como vimos que a classe NP tem problemas completos, podemos provar que cada level da hierarquia tem seu próprio problema completo. Uma linguagem L é Σ_k^p-completa se e somente se $L \in \Sigma_k^p$ e para todo $L' \in \Sigma_k^p$, $L' \leq_p L$.

Cada level da hierarquia polinomial tem a sua própria versão do problema SAT.

Definição 2.30. *Para todo $k > 0$, a linguagem Σ_k^pSAT consiste de todas as fórmulas lógicas ϕ tal que:*

$$\exists u_1 \in \{0,1\}^{p(|\langle\phi\rangle|)} \forall u_2 \in \{0,1\}^{p(|\langle\phi\rangle|)} \exists \dots Q_k u_k \in \{0,1\}^{p(|\langle\phi\rangle|)} \phi(u_1, u_2, \dots, u_k)$$

é verdadeiro, onde Q_k é $\exists$ se k é ímpar e $\forall$ se k é par.

Da mesma forma, uma fórmula lógica está em Π_k^pSAT se e somente o seguinte predicato quantificado é verdadeiro:

$$\forall u_1 \in \{0,1\}^{p(|\langle\phi\rangle|)} \exists u_2 \in \{0,1\}^{p(|\langle\phi\rangle|)} \forall \dots Q_k u_k \in \{0,1\}^{p(|\langle\phi\rangle|)} \phi(u_1, u_2, \dots, u_k)$$

Onde Q_k é $\forall$ se k é ímpar e $\exists$ se k é par.

E como é de se esperar, cada Σ_k^pSAT é Σ_k^p-completo. A prova que Π_k^pSAT é Π_k^p-completo, para cada $k \geq 1$, é análoga.

Teorema 2.31. *Para todo $k \geq 1$, Σ_k^pSAT é Σ_k^p-completo.*

Demonstração. Para algum $k \geq 1$, seja L uma linguagem em Σ_k^p. Para toda string $x' \in \{0,1\}^*$, nós temos que mostrar que existe uma redução em tempo polinomial que transforma x' em uma instância $\phi_{x'}$ de $\Sigma_k^p\mathsf{SAT}$ tal que $\phi_{x'} \in \Sigma_k^p\mathsf{SAT} \iff x' \in L$. Sabemos que podemos expressar L como:

$$x \in L \iff \exists x_1 \in \{0,1\}^{p(|x|)} \forall x_2 \in \{0,1\}^{p(|x|)} \exists \ldots Q_k x_k \in \{0,1\}^{p(|x|)} M(x, x_1, x_2, \ldots, x_k) = 1$$

Usando a redução do teorema 2.25, nós podemos transformar a máquina de Turing M em uma fórmula ϕ tal que $\phi(x, x_1, x_2, \ldots, x_k) = 1 \iff M(x, x_1, x_2, \ldots, x_k) = 1$ em tempo polinomial. Para cada $x' \in \{0,1\}^*$ nós criamos a fórmula $\phi_{x'}(x_1, x_2, \ldots, x_k) = \phi(x', x_1, x_2, \ldots, x_k)$ e temos que

$$\exists x_1 \in \{0,1\}^{p(|x|)} \forall x_2 \in \{0,1\}^{p(|x|)} \exists \ldots Q_k x_k \in \{0,1\}^{p(|x|)} \phi_{x'}(x_1, x_2, \ldots, x_k)$$

é verdade se e somente se

$$\exists x_1 \in \{0,1\}^{p(|x|)} \forall x_2 \in \{0,1\}^{p(|x|)} \exists \ldots Q_k x_k \in \{0,1\}^{p(|x|)} M(x', x_1, x_2, \ldots, x_k)$$

também é verdade. Ou seja, $\phi_{x'} \in \Sigma_k^p\mathsf{SAT}$ se e somente se $x' \in L$, como queriamos mostrar.

□

Complexidade de espaço

Além de tempo, uma outra medida de complexidade de máquinas de Turing importante é o número de células da fita de trabalho utilizadas, o que denominamos por complexidade de espaço. Analogamente a como fizemos para complexidade de tempo, nós definimos classes de complexidade que agrupam problemas de acordo com a sua complexidade de espaço. Ao definir classes de complexidade para problemas que usam menos do que $S(n)$ células nós iremos assumir que a função $S(n)$ é espaço construtível, que é definidada de forma análoga a como definimos funções tempo-construtível.

Definição 2.32 (Funções espaço-construtíveis)**.** *Uma função $f : \mathbb{N} \to \mathbb{N}$ é dita ser espaço-construtível se existe uma máquina de Turing M_f tal que ao receber a string unária 1^n em sua entrada, M_f escreve a representação binária de $f(n)$ em sua fita de saída e utiliza $\mathcal{O}(f(n))$ células da sua fita de trabalho.*

De novo temos que todas as funções que iremos usar ao longo deste trabalho para descrever a complexidade de espaço de uma máquina de Turing são espaço-contrutíveis. Então definimos classes de complexidade de espaço da seguinte forma.

Definição 2.33 (Classes de complexidade de espaço)**.** *Para uma função (espaço-construtível) $S : \mathbb{N} \to \mathbb{N}$ nós definimos as seguintes classes de complexidade de espaço.*

- $\mathsf{SPACE}(S(n))$*: a classe de todas as linguangens L tais que existe uma máquina de Turing determinística M que decide L em complexidade de espaço $S(n)$.*
- $\mathsf{NSPACE}(S(n))$*: a classe de todas as linguagens L tais que existe uma máquina de Turing não-determinística N que decide L e todos os possíveis ramos da computação de N computam em complexidade de espaço $S(n)$.*
- $\mathsf{coSPACE}(S(n))$*: uma linguagem L é dita estar em $\mathsf{coSPACE}(S(n))$ se e somente se a linguagem $\overline{L}$ está em $\mathsf{SPACE}(S(n))$. Também podemos definir $\mathsf{coNSPACE}(S(n))$ de forma similar.*

Nós então podemos definir a classe de problemas com complexidade de espaço polinomial.

Definição 2.34 (A classe PSPACE)**.** *A classe* PSPACE *é* $\bigcup_{c>0} \mathsf{SPACE}(n^c)$*. Ou seja, uma linguagem* L *está em* PSPACE *se e somente se* $L \in \mathsf{SPACE}(n^c)$*, para algum* $c > 0$*.*

A classe PSPACE é a versão da classe P para complexidade de espaço. Da mesma forma também temos a versão NP de complexidade de espaço que é a classe NPSPACE.

Definição 2.35 (A classe NPSPACE)**.** $\mathsf{NPSPACE} = \bigcup_{c>0} \mathsf{NTIME}(n^c)$.

Nós temos que $P \subseteq \mathsf{PSPACE}$ pois uma máquina de Turing que para após um número polinomial de passos só pode ter usado um número polinomial de células da sua fita de trabalho. Indo na outra direção, temos que $\mathsf{PSPACE} \subseteq \bigcup_{c>0} \mathsf{DTIME}(2^{n^c})$ (mais pra frente iremos ver que isto pode ser denotado como $\mathsf{PSPACE} \subseteq \mathsf{EXP}$) pelo seguinte argumento. Cada passo de uma máquina de Turing que usa apenas um número polinomial de células da sua fita de trabalho pode ser descrito por uma configuração usando um número polinomial de bits que iremos denotar por n^c, para algum $c > 0$. No entanto, por se tratar de uma máquina de Turing determinística temos que se ela repetir alguma das possíveis 2^{n^c} configurações ela necessariamente entrou em um loop infinito. Como estamos considerando máquinas de Turing que param após um número finito de passos temos que isto não pode acontecer e portanto há de ser o caso que a máquina de Turing pare após no máximo 2^{n^c} passos.

Da mesma forma temos que $\mathsf{NP} \subseteq \mathsf{PSPACE}$ pois podemos reutilizar a fita de trabalho para simular cada escolha não-determinística, e também $\mathsf{NPSPACE} \subseteq \bigcup_{c>0} \mathsf{SPACE}(2^{n^c})$ (equivalentemente, $\mathsf{NPSPACE} \subseteq \mathsf{EXP}$) pelo mesmo argumento que usamos para mostrar que $\mathsf{PSPACE} \subseteq \bigcup_{c>0} \mathsf{DTIME}(2^{n^c})$. Na verdade, no caso geral temos que

$$\mathsf{DTIME}(S(n)) \subseteq \mathsf{NTIME}(S(n)) \subseteq \mathsf{SPACE}(S(n)) \subseteq \mathsf{NSPACE}(S(n)) \subseteq \mathsf{DTIME}(2^{\mathcal{O}(S(n))}). \tag{2.9}$$

O fato que máquinas de Turing podem reutilizar as células de sua fita de trabalho pode ser usada para generalizar a afirmação $\mathsf{NP} \subseteq \mathsf{PSPACE}$ para $\mathsf{PH} \subseteq \mathsf{PSPACE}$.

Teorema 2.36. $\mathsf{PH} \subseteq \mathsf{PSPACE}$.

Demonstração. Seja $L \in \Sigma_k^p$, para algum $k > 0$. temos então que existe uma máquina de Turing M que roda em tempo $p(n)$ tal que L pode ser descrita da seguinte forma. Para todo $x \in \{0,1\}^n$,

$$x \in L \iff \exists z_1 \in \{0,1\}^{p(n)} \forall x_2 \in \{0,1\}^{p(n)} \dots Q_k z_k \in \{0,1\}^{p(n)} M(x, z_1, z_2, \dots, z_k) = 1.$$

Então podemos construir uma máquina de Turing que usa não mais do que $\mathcal{O}(p(n))$ células de sua fita de trabalho que verifica se existe alguma escolha z_1' para a string z_1 tal que

$$\exists z_1' \in \{0,1\}^{p(n)} \forall x_2 \in \{0,1\}^{p(n)} \dots Q_k z_k \in \{0,1\}^{p(n)} M(x, z_1, z_2, \dots, z_k) = 1.$$

Para cada escolha de z_1, M' pode reutilizar o espaço de sua fita de trabalho então o espaço que M' usa ao todo é simplesmente o espaço que ela usa para uma única escolha de z_1. O que basicamente estamos fazendo é descrever um procedimento recursivo, então podemos desenrolar a recursão e ver que o espaço utilizado é o espaço necessário para guardar as atribuições para as strings $z_1, z_2, \dots, z_k$ mais o espaço para simular a execução de M. Temos que ambos o termo são $\mathcal{O}(p(n)$ e portanto M' tem complexidade de espaço $\mathcal{O}(p(n))$. Portanto podemos concluir que $L \in \mathsf{PSPACE}$.

□

Nós podemos usar o argumento acima para argumentar que a seguinte linguagem está em PSPACE.

Definição 2.37. *Seja* TBQF *a linguagem que contém todas as fórmulas livres de quantificadores ϕ tais que*

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n \phi(x_1, x_2, \dots, x_n) = 1,$$

onde cada Q_i é $\exists$ ou $\forall$.

Não só temos que TBQF está em PSPACE mas também é verdade que TBQF é PSPACE-completa.

Definição 2.38 (Linguagens PSPACE-completas)**.** *Uma linguagem L é dita ser* PSPACE*-difícil se existe uma máquina de Turing R que roda em tempo polinomial e para toda linguagem $L' \in$ PSPACE,*

$$x \in L' \iff R(x) \in L.$$

Se também for o caso que $L \in$ PSPACE então dizemos que L é PSPACE*-completa.*

Então agora mostramos que TBQF é PSPACE-completa. Em 2.36 nós vimos que TBQF $\in$ PSPACE então agora basta mostrar que TBQF é PSPACE-difícil. Ou seja, só precisamos mostrar que para toda linguagem $L \in$ PSPACE, L é redutível em tempo polinomial para TBQF.

Teorema 2.39. *A linguagem* TBQF *é* PSPACE*-completa.*

Demonstração. □

Nós definimos complexidade de espaço como sendo a quantidade de células da *fita de trabalho* que uma máquina de Turing utliza durante a sua computação. Portanto, nada nos impede de definir classes de complexidade para funções sublineares. Por exemplo, uma classe de complexidade importante é a classe de todos os problemas com complexidade de espaço logaritmico.

Definição 2.40 (As classes Ł e NL.)**.** *Uma linguagem L é dita estar em Ł se existe uma máquina de Turing M com complexidade de espaço $\mathcal{O}(\log n)$ que decide L. Ou seja, Ł = $\mathsf{SPACE}(\log n)$.*

Uma linguagem L é dita estar em NL *se existe uma máquina de Turing N com complexidade de espaço não-determinística $\mathcal{O}(\log n)$ que decide L. Ou seja,* NL = $\mathsf{NSPACE}(\log n)$.

A partir de 2.9 podemos afirmar que Ł $\subseteq$ NL $\subseteq$ $\mathsf{SPACE}(2^{\mathcal{O}(\log n)}) =$ P. Ainda é um problema em aberto se P $\subseteq$ NL o que implicaria em P = NL [1].

Nós podemos definir linguagens NL-completas usando o seguinte tipo de reduções.

Definição 2.41 (Reduções logspace)**.** *Seja L, L'. Nós dizemos que L é logspace-redutível à L' se existe uma redução f tal que $x \in L \iff f(x) \in L'$ e além disso, existe uma máquina de Turing R com complexidade de espaço $\mathcal{O}(\log n)$ que satisfaz o seguinte. Para todo $\langle x, i\rangle$ em que $x \in \{0,1\}^*$ e $i \in \mathbb{N}$, ao receber $\langle x, i\rangle$ em sua entrada, R:*

1. *Rejeita a entrada se $i > |f(x)|$.*
2. *Dá como saída o bit $f(x)_i$ e mais um bit indicando que a entrada $\langle x, i\rangle$ não foi rejeitada caso $1 \leq i \leq |f(x)|$.*

[1] Também não se sabe se Ł = NL, no entanto o teorema de Savitch [Sav70] afirma que $\mathsf{SPACE}(S(n)) = \mathsf{NSPACE}(S(n)^2)$ para funções $S(n) \geq \log n$, o que implica em, por exemplo, PSPACE = NPSPACE.

Nós podemos provar que se redefinirmos problemas NP-completos usando reduções logspaces então a linguagem SAT continua sendo NP-completa. A idéia é usar o fato que a redução de Cook-Levin é estruturada de maneira bem repetitiva, então podemos calcular a partir do tamanho da entrada x da redução o bit que aparece na i-ésima fórmula que resultou da redução se escolhermos uma codificação apropriada para a fórmula.

Proposição 2.42 (SAT é NP-completa com respeito a reduções logspace)**.** *Vamos dizer que uma linguagem L' é* NP*-completa se $L' \in \mathsf{NP}$ e toda linguagem $L \in \mathsf{NP}$ é logspace-redutível à L'. Então, sob esta definição de* NP*-completude a linguagem* SAT *é* NP*-completa.*

Demonstração. □

Classes de complexidade de tempo exponencial

Teoremas de hierarquia

Como foi dito na introdução, um dos principais desafios da complexidade computacional é demonstrar limites inferiores para problemas computacionais. Intuitivamente podemos acreditar que problemas intrinsecamente difíceis devem existir pois se dermos mais recursos para uma máquina de Turing computar deveriamos também sermos capaz de decidir mais linguagens. Os teoremas de hierarquia é uma série de teoremas que provam exatamente isso. Geralmente, iremos usar o fato que máquinas de Turing que usam mais tempo podem simular máquinas de Turing que usam menos tempo para montar uma máquina de Turing "diagonalizadora", e então mostramos que esta máquina deve discordar com todas as máquinas que usam menos tempo em pelo menos um ponto.

Teorema 2.43. *(Teorema da hierarquia de tempo determinístico [HS65])*

Para todas funções $f, g : \mathbb{N} \to \mathbb{N}$ tempo-construtíveis satisfazendo $g(n)\log g(n) = o(f(n))$, temos que $\mathsf{DTIME}(g(n)) \subsetneq \mathsf{DTIME}(f(n))$.

Demonstração. Seja $L \in \mathsf{DTIME}(g(n))$, note que pelo teorema 2.14 existe uma máquina de Turing que simula a execução de uma máquina de Turing que decide L em tempo $\mathcal{O}(g(n)\log g(n)) = o(f(n))$. Então temos que deve existir uma máquina de Turing $\mathcal{D}$ que funciona da seguinte maneira:

- Sobre a entrada $\langle M \rangle$:
- Simule M sobre a entrada $\langle M \rangle$ por $g(|\langle M \rangle|)$ passos.
- Se em algum momento M aceita, rejeite a entrada; caso contrário, aceite.

Note que $\mathcal{D}$ roda em tempo $\mathcal{O}(g(n)\log g(n))$ e portanto $L(\mathcal{D}) \in \mathsf{DTIME}(f(n))$.

Nós afirmamos que $L(\mathcal{D}) \notin \mathsf{DTIME}(g(n))$. Assuma o contrário, que $L(\mathcal{D}) \in \mathsf{DTIME}(g(n))$ e chame de $\mathcal{D}'$ uma máquina de Turing que decida $L(\mathcal{D})$ em tempo $g(n)$. Então usamos o mesmo argumento que usamos para mostrar que HALT não é decidível para mostrar que $L(\mathcal{D}) \notin \mathsf{DTIME}(g(n))$ (ou equivalentemente, que tal máquina de Turing $\mathcal{D}'$ não pode existir): nós rodamos $\mathcal{D}'$ sobre a sua própria descrição e vemos o que acontece.

- Se $\mathcal{D}'(\langle \mathcal{D}' \rangle) = 1$ então $\mathcal{D}$ pela sua definição deve rejeitar a entrada $\langle \mathcal{D}' \rangle \Rightarrow \mathcal{D}(\langle \mathcal{D}' \rangle) = 0$
- Se $\mathcal{D}'(\langle \mathcal{D}' \rangle) = 0$ então $\mathcal{D}$ aceita a entrada $\langle \mathcal{D} \rangle \Rightarrow \mathcal{D}(\langle \mathcal{D}' \rangle) = 1$

Nos dois itens acima nós temos que assumir que a descrição de $\mathcal{D}'$ seja grande o suficiente para que D possa simular a toda a tua execução sobre a entrada $\langle \mathcal{D}' \rangle$, mas isso não é problema já que D' possui descrições arbitrariamente grandes.

Ambos os casos são contradições e portanto temos que $L(\mathcal{D}) \notin \mathsf{DTIME}(g(n))$ e $\mathsf{DTIME}(g(n)) \subsetneq \mathsf{DTIME}(f(n))$.

□

Agora nós vamos considerar o caso não-determinístico. Se tentarmos provar da mesma forma um teorema de hierarquia de tempo não-determinístico nós esbarrariamos no seguinte problema: não é óbvio o modo como podemos negar uma computação não-determinística, pois deveriamos ter um conhecimento "universal" sobre todas os ramos da computação. A única forma óbvia de fazer isso seria simular todos os ramos da computação, o que leva tempo exponencial. Porém, ainda pode-se usar essa simulação determinística para diagonalizar máquinas de Turing não-determinística, dado que a simulação seja feita numa entrada exponencialmente menor do que a entrada original. O truque é fazer que uma máquina "diagonalizadora" $\mathcal{D}$ ou discorde de uma máquina de Turing não-determinística M em alguma string unária de tamanho em um intervalo de comprimento exponencial ou que ela discorde nos extremos deste intervalo. Na prova do teorema abaixo nós usamos o fato que uma simulação não-deterministica pode ser feita com "slowdown" constante.

Teorema 2.44. *(Teorema da hierarquia de tempo não-determinístico [Coo73])*

Sejam $f, g : \mathbb{N} \to \mathbb{N}$ funções tempo-construtíveis satisfazendo $g(n+1) = o(f(n))$, então $\mathsf{NTIME}(g(n)) \subsetneq \mathsf{NTIME}(f(n))$.

Demonstração. Nesta prova, $\{M_i\}_{i\in\mathbb{N}}$ representa uma enumeração de todas as máquinas de Turing não-determinísticas.

Considere a função h definida como $h(1) = 2$ e $h(i+1) = 2^{g(h(i)+1)}$, para $i > 1$. Nós construimos uma máquina de Turing não-determinística $\mathcal{D}$ que inicialmente assumismo concordar com M_i em todas as strings unárias 1^n satisfazendo $h(i) < n \leq h(i+1)$, daí ela diagonaliza na entrada $1^{h(i+1)}$ e a partir disso nós obtemos uma contradição. Como $h(i+1)$ é exponencialmente maior do que $h(i)+1$, $\mathcal{D}$ pode simular todos os ramos da computação de M_i sobre a entrada $1^{h(i)+1}$ deterministicamente. A máquina de Turing $\mathcal{D}$ é definida da seguinte maneira (qualquer entrada que não seja da forma 1^n para $n \in \mathbb{N}$ é imediatamente rejeitada):

1. Sobre a entrada 1^n, ache i tal que $h(i) < n \leq h(i+1)$.
2. Se $h(i) < n < h(i+1)$, $\mathcal{D}$ não-deterministicamente simula M_i sobre a entrada 1^{n+1} por $g(n+1)$ passos e aceita se e somente se M_i aceita.
3. Se $n = h(i+1)$, $\mathcal{D}$ deterministicamente simula M_i sobre a entrada $1^{h(i)+1}$ por $g(h(i)+1)$ passos e aceita se e somente se M_i rejeita.

Primeiro, nós temos que $\mathcal{D}$ roda em tempo não-determinístico $\mathcal{O}(f(n))$, pois estamos assumindo que $g(n+1) = o(f(n))$. Vamos assumir que exista uma máquina de Turing não-determinística $\mathcal{D}'$ que roda em tempo $\mathcal{O}(g(n))$ e $L(\mathcal{D}') = L(\mathcal{D})$. Seja i suficientemente grande tal que $\mathcal{D}' = M_i$. Se existe algum $h(i) < n \leq h(i+1)$ tal que $\mathcal{D}'(1^n) \neq \mathcal{D}(1^n)$ então estamos feitos. Caso contrário, considere a entrada $1^{h(i+1)}$. Nós temos o seguinte.

- Se $\mathcal{D}'(1^{h(i)+1}) = 1$ então $\mathcal{D}$ deve rejeitar a entrada $1^{h(i+1)}$.
- Se $\mathcal{D}'(1^{h(i)+1}) = 0$ então $\mathcal{D}$ deve aceitar a entrada $1^{h(i+1)}$.

Então se quisermos evitar uma contradição precisamos ter $\mathcal{D}'(1^{h(i)+1}) \neq \mathcal{D}(1^{h(i+1)})$. Mas temos pelo item (2) na descrição de $\mathcal{D}$ e pela nossa suposição que

$$\mathcal{D}'(1^{h(i)+1}) = \mathcal{D}(1^{h(i)+2}) = \mathcal{D}'(1^{h(i)+2}) = \mathcal{D}(1^{h(i)+3}) = \cdots = \mathcal{D}'(1^{h(i+1)-1}) = \mathcal{D}(1^{h(i+1)}).$$

E portanto temos uma contradição.

□

Limites da diagonalização

Podemos nos perguntar se a estratégia de diagonalização usadas nas provas dos teoremas de hierarquia pode também nos dar limites inferiores mais interessantes. Podemos separar as classes P e NP usando diagonalização?

Vamos imaginar que nós temos uma prova que $\mathsf{P} \neq \mathsf{NP}$ que usa diagonalização da forma que usamos para provar o teoremas de hierarquia. Ou seja, nós temos uma máquina de Turing $\mathcal{D}$ construida de forma que ela difere de todas as máquina de Turing M "em P" em pelo menos um ponto, simulando a execução de M e depois invertendo a saída de M sobre alguma entrada. Como podemos também enumerar e simular máquinas de Turing com qualquer oráculo $\mathcal{O}$ da mesma forma que podemos enumerar e simular máquinas de Turing convencionais, ao adicionar o oráculo $\mathcal{O}$ à $\mathcal{D}$ nós podemos também separar $\mathcal{D}^{\mathcal{O}}$ de todas as outras máquinas de Turing em $\mathsf{P}^{\mathcal{O}}$ de forma análoga. Portanto, uma prova que $\mathsf{P} \neq \mathsf{NP}$ que usa diagonalização deve também provar que $\mathsf{P}^{\mathcal{O}} \neq \mathsf{NP}^{\mathcal{O}}$. Porém, o objetivo desta seção é mostrar que existem oráculos $\mathcal{A}$ e $\mathcal{B}$ tal que $\mathsf{P}^{\mathcal{A}} = \mathsf{NP}^{\mathcal{A}}$ e $\mathsf{P}^{\mathcal{B}} \neq \mathsf{NP}^{\mathcal{B}}$.

Teorema 2.45. *Existem oráculos $\mathcal{A}$ e $\mathcal{B}$ tais que:*

1. $\mathsf{P}^{\mathcal{A}} = \mathsf{NP}^{\mathcal{A}}$.
2. $\mathsf{P}^{\mathcal{B}} \neq \mathsf{NP}^{\mathcal{B}}$.

Demonstração. Para provar 1 nós fazemos $\mathcal{A}$ ser um oráculo para a linguagem PSPACE-completa TBQF. Daí, por simulações, nós temos que

$$\mathsf{P}^{\mathsf{TBQF}} \subseteq \mathsf{NP}^{\mathsf{TBQF}} \subseteq \mathsf{PSPACE},$$

e por redução,

$$\mathsf{PSPACE} \subseteq \mathsf{P}^{\mathsf{TBQF}}$$

O oráculo B nós vamos contruí-lo de forma que nenhuma máquina de Turing que executa menos do que $2^n/10$ passos possa decidir a linguagem $L_{\mathcal{B}}$ definida como

$$L_{\mathcal{B}} = \{1^n | \exists y \in \mathcal{B} \text{ tal que } |y| = n\}$$

Note que $L_{\mathcal{B}} \in \mathsf{NP}^{\mathcal{B}}$.

Seja $\{M_i\}_{i\in\mathbb{N}}$ uma enumeração de todas máquinas de Turing de tempo polinomial e $\{p_i\}_{i\in\mathbb{N}}$ o tempo de execução de cada uma das $M_i s$.

Para M_1, escolhemos um número n_1 grande o suficiente de forma que $p_1(n_1) < 2^{n_1}$. Seja $q_1, q_2, \ldots, q_l$, com $l < p_1(n_1)$, as strings consultadas por M_1 na entrada 1^{n_1} e declaramos que cada $q_i \notin \mathcal{B}$. Então,

- Se $M_1(1^{n_1}) = 1 \Rightarrow$ declare todas as strings de tamanho n_1 como não estando em $\mathcal{B}$.
- Se $M_1(1^{n_1}) = 0 \Rightarrow$ declare pelo menos uma string não consultada por M_1 como estando em $\mathcal{B}$.

Como $p_1(n_1) < 2^{n_1}$ temos que tal string deve existir.

Para $i > 1$ nós fazemos a mesma coisa. Nós definimos n_i como sendo o menor número em $\mathbb{N}$ tal que $p_i(n_i) < 2^{n_i}$ e $n_i > n_{i-1}$. Como podemos assumir que nos passos anteriores nenhuma string de tamanho maior do que n_{i-1} já esteve sua pertinência em $\mathcal{B}$ resolvida, podemos proceder da mesma forma como fizemos no primeiro estágio.

Então, para cada M_i nós descrevemos uma string 1^{n_i} tal que $M_i(1^{n_i}) = 1 \iff 1^{n_i} \notin L_{\mathcal{B}}$. Portanto, nenhuma das máquinas de Turing de tempo polinomial M_i é um decisor para $L_{\mathcal{B}}$ e consequentemente $L_{\mathcal{B}} \notin \mathsf{P}^{\mathcal{B}}$.

□

2.6 Complexidade de circuitos

O *tamanho* de um circuito booleano C é o número de portas lógicas que o compõem e a sua profundidade é o maior caminho de uma das variáveis de entrada até a saída de C. Denotamos o tamanho de C por $|C|$.

Definição 2.46. *(Medição de complexidade de circuitos)*

Para funções $f, g : \mathbb{N} \to \mathbb{N}$ dizemos que uma linguagem L é dita estar em $\mathsf{SIZE}(f(n))$ se existe uma família de circuitos $\{C_n\}_{n\in\mathbb{N}}$ tal que $|C_n| \leq f(n)$, para todos n, e que L é dita estar em $\mathsf{DEPTH}(g(n))$ se a profundidade de C_n é menor ou igual a $g(n)$, para todos n.

Uma observação quanto a representação de circuitos booleanos por strings. Como um circuito é um grafo não direcionado acíclico, nós podemos representá-lo pela sua lista de adjacência. A lista de adjacência de um circuito de tamanho S tem S linhas onde cada linha irá conter:

- O label da porta lógica.
- O índice das portas lógicas que alimentam a entrada desta porta lógica.

Se assumirmos que o fan-in de todas portas lógicas é limitado por uma constante então precisamos de $c \log S$ bits para formar uma das linhas da lista de adjacência de C, para alguma constante c, portanto ao todo precisamos de $\mathcal{O}(S \log S)$ bits para representar um circuito pela sua lista de adjacência.

Algumas vezes nós vamos querer usa máquinas de Turing para simular a computação de um circuito C sobre a entrada x. Considere o seguinte problema:

Definição 2.47 (Problema da avaliação de circuito)**.** *O problema da avaliação de circuito, que nós chamaremos de* CIRCUIT-EVAL*, consiste de todos pares $\langle C, x\rangle$ onde C é um circuito booleano e x uma string binária tal que $C(x) = 1$.*

Podemos verificar que existe uma máquina de Turing que decide CIRCUIT-EVAL em tempo linear usando basicamente o procedimento descrito na definição 2.7 em que avaliamos o valor de cada porta lógica de C seguindo um ordenamento topológico.

Circuitos de tamanho polinomial

Assim como fizemos na seção anterior, nós queremos definir uma classe que procura capturar todas as linguagens que são decididas por circuitos "pequenos". Assim como fizemos com máquinas de Turing, nós usamos complexidade polinomial como sinônimo de eficiência.

Definição 2.48 (P/poly)**.** *Uma linguagem L é dita estar em* P/poly *se e somente se existe $c > 0$ tal que $L \in \mathsf{SIZE}(n^c)$.*

Ou seja, $L \in \mathsf{P/poly}$ se e somente se existe uma família de circuitos $\{C_n\}_{n\in\mathbb{N}}$ que decide L onde todos os circuitos em $\{C_n\}_{n\in\mathbb{N}}$ têm tamanho polinomial.

Nós sabemos que vários problemas úteis têm circuitos pequenos, como adição e outras operações aritméticas. Na verdade, todos problemas que podem ser resolvidos eficientemente por uma máquina de Turing tem um famílias de circuito de tamanho polinomial como mostra o teorema a seguir.

Teorema 2.49. $\mathsf{P} \subseteq \mathsf{P/poly}$.

Podemos provar este teorema mostrando que para todas máquinas de Turing que param em menos do que $T(n)$ passos existe um circuito de tamanho $\mathcal{O}(T(n) \log T(n))$.

Demonstração. Seja L uma linguagem em $\mathsf{DTIME}(T(n))$. Nós sabemos que existe uma máquina de Turing oblivious $A = (\Gamma, Q, \delta)$ de duas fitas que computa L em menos do que $T'(n) = \mathcal{O}(T(n) \log T(n))$ passos. Nós podemos usar A para construir circuitos C_n de tamanho $\mathcal{O}(T'(n))$ que decidem L restrita à strings de tamanho n, para todos $n > 0$. C_n tem $T'(n)$ níveis em que cada nível tem um número constante de portas lógicas e portanto $|C_n| = \mathcal{O}(T'(n)) = \mathcal{O}(T(n) \log T(n))$.

Cada nível de C_n é composto por um subcircuito que computa a função de transição de A. Chamaremos este subcircuito de C_δ e nós escreveremos C_δ^i quando queremos especificar o subcircuito que se encontra no i-ésimo nível. O que nós queremos é que o i-ésimo nível de C_n compute a configuração que A entra no $(i+1)$-ésimo passo. As entradas do subcircuito C_δ é dividida em três partes que representam o estado atual e os símbolos sendo lidos pelos dois cabeçotes de A. No total a entrada tem $\log|Q| + 3$ bits.[2] As saídas são particionada em duas partes que representam o estado de A no próximo passo e o símbolo escrito na fita de trabalho. O estado na entrada de C_δ^i é o estado na saída de C_δ^{i-1}, o símbolo na fita de entrada de C_δ^i vem de uma das entradas de C_n e o símbolo na fita de trabalho vem da saída de C_δ^j, onde j é o maior número menor do que i tal que no passo j o cabeçote da fita de trabalho esteve na mesma posição em que ele se encontra no passo i. A entrada de C_δ^1 é o estado inicial de A e caso o "j" não exista então o símbolo sendo lido na fita de trabalho é $\square$, ambos podem ser implementados usando as constantes '1' e '0' usando um número constante de portas lógicas. Como C_δ depende somente da função de transição de A temos que $|C_\delta| = \mathcal{O}(1)$.

$\square$

O problema CIRCUIT-SAT

No problema CIRCUIT-SAT é dado um circuito C com n entradas e queremos decidir se existe $x \in \{0,1\}^n$ tal que $C(x) = 1$. Dizendo de outra maneira, queremos decidir se C computa ou não a função constante $C(x) = 0$. A seguir nós vemos que CIRCUIT-SAT é NP-completo e portanto ainda não se sabe se existe um algoritmo eficiente para computá-lo.

Teorema 2.50. CIRCUIT-SAT *é* NP*-completo.*

Demonstração. CIRCUIT-SAT está em NP pois uma atribuição às variáveis de entrada de C servem como certificado.

Se $L \in \mathsf{NP}$ então existe uma máquina de Turing oblivious M de tempo polinomial e polinômio p tal que $M(x,u) = 1$ sempre que $x \in L$ e $u \in \{0,1\}^{p(|x|)}$ é um certificado para x. Daí, pela construção na prova do teorema 2.49 nós podemos construir um circuito C que computa $M_x(u) = M(x,u)$. C é satisfazível se e somente se existe u tal que $M(x,u) = 1$, que é o mesmo que dizer que $x \in L$.

A redução pode ser feita em tempo polinomial pois para construir cada nível do circuito nós precisamos somente da descrição de C_δ e a posição dos cabeçotes nas duas fitas de M_x. Para decidir a posição dos cabeçotes de M_x no passo i em tempo polinomial nós simplesmente simulamos M_x sobre uma entrada de tamanho $p(|x|)$ por i passos.

$\square$

[2] Para representar o símbolo sendo lido na fita de entrada precisamos de somente um bit, para representar o símbolo sendo lido pela fita de trabalho precisamos de um bit adicional porque este símbolo pode ser $\square$.

Famílias de circuitos uniforme

Famílias de circuitos como vimos até agora são um modelo de computação meio que irrealista. Tome por exemplo a linguagem HALT. Não só existe uma família de circuito que decide HALT mas também temos que HALT está em P/poly se modificarmos ela um pouco. É só considerar a redução de HALT para a linguagem unária $\{1^n |$ a n-ésima string binária na ordem lexicográfica pertence a HALT$\}$, que chamamos de UHALT. Todas linguagens unárias $\mathcal{U}$ são decididas por uma família de circuito de tamanho polinomial: para cada n tal que $1^n \notin \mathcal{U}$, C_n é simplesmente o circuito que computa a função constante 0, e se $1^n \in \mathcal{U}$ então C_n computa o mintermo $x_1 \wedge \cdots \wedge x_n$. Portanto UHALT $\in$ P/poly mesmo ela sendo claramente indecidível (isto é, não existe nenhuma máquina de Turing que a decida).

Para contornar este problema podemos exigir que cada circuito de uma família de circuitos seja computável de alguma forma. Além disso, podemos também exigir que eles sejam eficientemente computáveis como veremos a seguir.

Definição 2.51. *(Família de circuitos* P*-uniforme)*

Uma família de circuitos $\{C_n\}_{n\in\mathbb{N}}$ é dita ser P*-uniforme sse existe uma máquina de Turing A de tempo polinomial tal que sobre a entrada 1^n, A dá como saída o circuito C_n.*

E nós podemos provar que a classe de linguagens decididas por famílias de circuitos P-uniforme coincide com P.

Teorema 2.52. *Uma linguagem L é computável por uma família de circuitos* P*-uniforme sse $L \in$ P.*

Demonstração.

- L é computável por uma família de circuitos P-uniforme $\Rightarrow L \in$ P:

  Seja A a máquina de turing que computa C_n a partir de 1^n. Nós contruimos uma máquina de turing B que sobre uma entrada x, simula A sobre a entrada $1^{|x|}$ para obter $C_{|x|}$. Daí podemos computar $C_{|x|}(x)$ em tempo polinomial.

- $L \in$ P $\Rightarrow L$ é computável por uma família de circuitos P-uniforme:

  Como $L \in$ P temos que existe uma máquina de Turing oblivious M que computa L em tempo $p(n) \log p(n)$ = poly$(p(n))$, para algum polinômio p. Usando a redução na prova do teorema 2.49, que sabemos que roda em tempo polinomial, podemos construir C_n a partir de 1^n.

□

Nós podemos melhorar o resultado no teorema 2.52 restrigindo ainda mais o recurso usado para computar C_n:

Definição 2.53 (Família de circuitos logspace-uniforme)**.** *Uma família de circuitos é dita ser logspace-uniforme sse existe uma máquina de Turing que computa C_n a partir de 1^n usando somente $\mathcal{O}(\log n)$ espaços de memória.*

Teorema 2.54. *Uma linguagem L é computada por uma família de circuitos logspace-uniforme sse $L \in$ P.*

Demonstração.

A ida é consequência de toda computação que usa espaço logaritmo poder ser feita em tempo polinomial.

Para provar a volta vamos mostrar que a redução na prova do teorema 2.49 pode ser feita em espaço logaritmo. Seja L uma linguagem em P e A uma máquina de Turing oblivious de duas fitas que decide L

em menos do que $p(n)$ passos, onde p é um polinômio. Nós queremos mostrar que existe uma máquina de Turing M que usa não mais do que $\mathcal{O}(\log n)$ espaços de memória que dado 1^n, M escreve em sua fita de saída a descrição do circuito C_n usando a construção na prova do teorema 2.49. Para construir C_n a partir de 1^n nós precisamos calcular $p(n)$, a descrição do circuito C_δ (δ é a função de transição de A) e também deve ser possível, dado um inteiro $i \leq p(n)$, calcular as células em que os dois cabeçotes de A se encontram no i-ésimo passo. Calcular $p(n)$ pode ser feito em espaço logaritmo e calcular a descrição de C_δ pode ser feito em tempo constante. Para decidir as posições do cabeçotes em um dado passo, nós simulamos A sem nos preocuparmos com o estado e os símbolos lidos para que não precisamos ter que guardar o conteúdo das fitas de A em algum lugar. Podemos ignorar o contéudo das fitas e o estado atual porque os cabeçotes se movimentam em função apenas do tamanho da entrada. Dessa forma só precisamos manter três contadores que contam o número de passos executados e as posições dos cabeçotes das duas fitas de A.

□

Teorema de Karp-Lipton

Uma das maiores motivações para o estudo de complexidade de circuitos é a perspectiva de podermos separar P e NP usando resultados nesta área. Nós já vimos que $\mathsf{P} \subseteq \mathsf{P/poly}$, portanto se provarmos que $\mathsf{NP} \nsubseteq \mathsf{P/poly}$ nós teremos provado que $\mathsf{P} \neq \mathsf{NP}$. Nós iremos ver agora que $\mathsf{NP} \subseteq \mathsf{P/poly}$ implica no colapso da hierarquia polinomial.

Lema 2.55. *Se* $\mathsf{SAT} \in \mathsf{P/poly}$ *então existe um polinômio* p *tal que para todo* $n > 0$ *existe uma máquina de Turing* M_n *que ao receber a descrição de uma fórmula* ϕ *satisfazível de tamanho* n*,* M_n *retorna um* y *tal que* $\phi(y) = 1$ *em menos do que* $p(n)$ *passos.*

Demonstração.

Seja $\{C_n\}_{n\in\mathbb{N}}$ uma família de circuitos de tamanho polinomial que decida SAT.

A máquina M_n tem a capacidade de escrever em uma das suas fitas de trabalho a descrição de C_k, $1 \leq k \leq n$. Ao receber $\langle\phi\rangle$ em sua fita de entrada, M_n primeiro verifica se $|\langle\phi\rangle| = n$, e se não for o caso que $|\langle\phi\rangle| = n$, M_n imediatemente rejeita a entrada.

Se $|\langle\phi\rangle| = n$ então primeiro verificamos se ϕ é satisfazível usando o circuito C_n, rejeitando a entrada $\langle\phi\rangle$ caso não seja. Daí, nós podemos achar uma atribuição $y \in \{0,1\}^m$ tal que $\phi(y) = 1$, m é o número de entradas de ϕ, decidindo sequencialmente o valor de cada variável de ϕ na atribuição y.

Primeiro faça $x_1 = 1$ e verifique se ϕ é satisfazível quando trocamos cada aparição de x_1 em ϕ pela constante 1. Se sim, então fixe $x_1 = 1$ e se não fixe $x_1 = 0$. Uma das duas restrições ($x_1 = 1$ ou $x_1 = 0$) deve manter ϕ satisfazível. Restringir o valor de algumas variáveis de ϕ resulta em uma fórmula menor, e por M_n poder escrever em sua fita de trabalho o circuito C_k, para todo $k \leq n$, podemos usar um destes circuitos para decidir a satisfazibilidade da fórmula restrita. Nós podemos continuar usando o mesmo procedimento que usamos para decidir o valor de x_1 para todas as outras variáveis, sempre diminuindo o tamanho do circuito. No fim nós teremos obtido a atribuição y.

Para ver que M_n roda em tempo menor do que $p(n)$, para algum polinômio p, nós simplesmente notamos que a operação mais custosa na computação de M_n é escrever a descrição de C_k, $1 \leq k \leq n$, leva tempo polinomial pois C_k tem tamanho polinomial.

□

Teorema 2.56 (Teorema de Karp-Lipton [KL82])**.** $\mathsf{NP} \subseteq \mathsf{P/poly} \Longrightarrow \mathsf{PH} = \Sigma_2^p$.

Demonstração.

Assuma que $\mathsf{NP} \subseteq \mathsf{P/poly}$.

Pelo teorema 2.29 sabemos que para mostrar que $\mathsf{PH} = \Sigma_2^p$, é suficiente mostrar que $\Pi_2^p \subseteq \Sigma_2^p$. E para isso nós mostramos que a linguagem Π_2^p-completa $\Pi_2^p\mathsf{SAT}$ está em Σ_2^p.

Lembrando que a linguagem $\Pi_2^p\mathsf{SAT}$ contém todas fórmulas ϕ tal que

$$\forall x \in \{0,1\}^n \ \exists y \in \{0,1\}^n \phi(x,y) \tag{2.10}$$

é verdadeira. Ou seja, para cada $x \in \{0,1\}^n$, fixar o primeiro parâmetro de ϕ mantém a fórmula ϕ satisfazível. Denotamos $\phi(x,.)$ por $\phi_x(.)$ e dizer que $\phi \in \Pi_2^p\mathsf{SAT}$ é equivalente a dizer que para todos $x \in \{0,1\}^n$, ϕ_x é satisfazível. Isso é o mesmo que dizer que para um circuito C que computa atribuições que satisfazem fórmulas satisfazíveis:

$$\forall x \phi_x(C(\langle\phi\rangle, x)) = 1$$

Já que $\mathsf{NP} \subseteq \mathsf{P/poly}$ implica em SAT ter circuitos de tamanho polinomial, pelo lema 2.55 temos que tal circuito C deve ter tamanho polinomial, então o seguinte é verdadeiro se assumirmos que ϕ_x é satisfazível para todo x (ou seja, $\phi \in \Pi_2^p\mathsf{SAT}$):

$$\exists C' \in \{0,1\}^{p(n)} \forall x \in \{0,1\}^n \phi_x(C'(\langle\phi\rangle, x)) = 1 \tag{2.11}$$

Onde p é um polinômio.

Se 2.10 é verdadeiro então o circuito C irá, para todas strings $x \in \{0,1\}^n$, apresentar em tua saída um $y \in \{0,1\}^n$ tal que $\phi_x(x,y) = 1$ e portanto 2.11 também é verdadeiro.

E por outro lado, se 2.11 é verdadeiro então para todo $x \in \{0,1\}^n$, ϕ_x é satisfazível e portanto 2.10 também é verdadeiro.

Ou seja, 2.11 é uma formulação equivalente do problema $\Pi_2^p\mathsf{SAT}$ que pode ser visto como uma linguagem em Σ_2^p.

□

Limitantes inferiores e superiores para tamanho de circuitos

Nós já vimos que usando a base $\Omega = \{\vee, \wedge, \neg\}$ nós podemos implementar qualquer função booleana de n entradas usando circuitos de tamanho $\mathcal{O}(n2^n)$ através de mintermos. E também acabamos de ver que acredita-se que algumas linguagens em NP exigem circuitos de tamanho superpolinomial.

Agora nós vamos ver que nem todas funções podem ser computadas por circuitos de tamanho polinomial. E também veremos um limite superior melhor do que $\mathcal{O}(n2^n)$. Na verdade, vamos ver que ambos o limites difereciam entre si por um fator constante.

O limite inferior foi descoberto por Shannon em dos primeiros resultados em complexidade de circuitos. Em 1949, Shannon estava interessado no problema de minimização de circuitos quando ele publicou o resultado que nos vamos ver em seguida em [S$^+$49], a prova que eu apresento aqui foi tirada de [AB09].

Teorema 2.57. *Para todo $n > 1$, existem uma constante d e funções booleanas $f : \{0,1\}^n \to \{0,1\}$ tal que f não é computada por circuitos de tamanho menor do que $\frac{2^n}{dn}$.*

Demonstração. Nós vimos que todos circuitos de tamanho S podem ser representados através de sua lista de adjacência usando no máximo $cS \log S$ bits, para alguma constante $c > 1$. Então, o número de circuitos de tamanho no máximo S é menor ou igual ao número de strings de tamanho $cS \log S$ que é igual a $2^{cS \log S}$. Por

outro lado, o número de funções booleanas de n entradas é 2^{2^n}. Fazendo $S = \frac{2^n}{dn}$, para alguma constante $d > c$, nós temos o seguinte:

$$\begin{aligned} 2^{cS\log S} &= 2^{c\frac{2^n}{dn}\log(\frac{2^n}{dn})} \\ &= 2^{\frac{c2^n}{dn}(n-\log dn)} \\ &< 2^{\frac{2^n cn}{dn}} \\ &< 2^{2^n} \end{aligned}$$

O número de funções booleanas de n entradas que têm circuitos de tamanho $\frac{2^n}{dn}$ é menor do que do que o número total de funções booleanas de n entradas. Portanto deve existir funções booleanas com n entradas que exigem circuitos com no mínimo $\frac{2^n}{dn}$ portas lógicas.

□

Nós também podemos notar que $\frac{2^{\frac{c}{d}2^n}}{2^{2^n}} = 2^{2^n(\frac{c}{d}-1)} = 2^{-\Theta(2^n)}$, o que significa que a fração de funções booleanas que têm circuitos de tamanho menor do que $\frac{2^n}{dn}$ é bem pequena.

Indo na mesma direção nós podemos nos perguntar se é possível obter um limitante inferior para o tamanho de circuitos que aproximem uma função f. Seja $f : \{0,1\}^n \to \{0,1\}$ uma função Booleana qualquer, nós dizemos que $g : \{0,1\}^n \to \{0,1\}$ aproxima f com vantagem $\varepsilon > 0$ se

$$\left| \Pr[f(x) = g(x)] - 1/2 \right| > \varepsilon.$$

Se tratarmos g como uma função aleatória, ou seja para cada $x \in \{0,1\}^n$ nós fazemos $g(x) = 1$ com probabilidade $1/2$ e $g(x) = 0$ com probabilidade $1/2$, e denotarmos por $Z_{f,x}$ a variável aleatória indicadora que é 1 se e somente se $f(x) = g(x)$, então usando a desigualdade de Hoeffding:

$$\Pr\left[2^{-n} \sum_{x\in\{0,1\}^n} \Big(Z_{f,x} - E[Z_{f,x}]\Big) \geq \varepsilon\right] \leq e^{-2\varepsilon^2 2^n}. \tag{2.12}$$

O que significa que com probabilidade muito alta uma função aleatória g vai falhar em aproximar f com vantagem maior do que ε. Usando a desigualdade acima podemos provar o segunte teorema.

Teorema 2.58. *Para todo $\varepsilon > 0$, existe uma constante d e função $f : \{0,1\}^n \to \{0,1\}$ tal que para todas funções $g : \{0,1\}^n \to \{0,1\}$ que admitem circuitos de tamanho no máximo $S = \frac{2^n}{dn}$ é verdade que*

$$\Pr[f(\boldsymbol{x}) \neq g(\boldsymbol{x})] \geq 1/2 + \varepsilon.$$

Demonstração. Seja c a constante tal que todo circuito de tamanho no máximo S pode ser representado por uma string de comprimento $cS\log S$ e para $\varepsilon > 0$ seja $d = d(\varepsilon) > \frac{c}{2log(e)\varepsilon^2}$.

Para todas funções $g : \{0,1\}^n \to \{0,1\}$ que admitem circuitos de tamanho menor do que $S = \frac{2^n}{dn}$ e para toda função Booleana $f : \{0,1\}^n \to \{0,1\}$, seja $Z_{f,g}$ a variável aleatória indicadora que é 1 se e somente se $\Pr[f(x) \neq g(x)] \geq 1/2 + \varepsilon$. Nós podemos notar que a desigualdade 2.12 implica em $E_{\boldsymbol{g}}[Z_{f,\boldsymbol{g}}] \geq 1 - e^{-2\varepsilon^2 2^n}$, quando fixamos uma função f arbitrária. Logo, pelo princípio da inclusão-exclusão a probabilidade que existe uma função f que satisfaz $Z_{f,g} = 1$ para todas funções g com circuitos de tamanho no máximo S é limitada por baixo por

$$\begin{aligned}
1 - 2^{cS\log S} e^{-2\varepsilon^2 2^n} &= 1 - 2^{cS\log S - 2\log(e)\varepsilon^2 2^n} \\
&= 1 - 2^{c\frac{2^n}{dn}(n - \log(dn)) - 2\log(e)\varepsilon^2 2^n} \\
&> 1 - 2^{2\log(e)\varepsilon^2 2^n - 2\log(e)\varepsilon^2 2^n} \\
&= 0.
\end{aligned}$$

Portanto, pelo método probabilístico, deve haver um g que falha em ser aproximada com vantagem ε por todos circuitos de tamanho no máximo S.

□

Os dois limitantes inferiores em 2.57 e 2.58 são bem precisos na verdade. Em [Lup58] Lupanov apresenta uma representação de funções booleanas que nos leva a um limite superior de $(1 + o(1))\frac{2^n}{n}$.

Definição 2.59 (Representação de Lupanov)**.** *Uma (k, s)-representação de Lupanov de uma fórmula booleana é descrita a seguir.*

- *Particione as n variáveis de f em duas partes:* $x_1, \ldots, x_k$ *e* $x_{k+1}, \ldots, x_n$
- *Construa uma matrix* $2^k \times 2^{n-k}$ *onde as linhas da matriz são indexadas pelas atribuições às k primeiras variáveis de f e as colunas são indexadas pelas atribuições às* $n - k$ *últimas variáveis. A entrada (x, y),* $x \in \{0,1\}^k$ *e* $y \in \{0,1\}^{n-k}$*, desta matriz é* $f(x, y)$*.*
- *Seja* $p = \lceil \frac{2^k}{s} \rceil$*, temos p conjuntos* A_i*,* $1 \leq i \leq p$*, onde* A_i *contém as s linhas* $(i-1)s$ *até* $is - 1$ *da matriz que nós construimos.* A_p *pode ter menos do que s linhas caso s não seja um divisor de* 2^k*.*

Daí nós definimos para cada $1 \leq i \leq p$ *e* $w \in \{0,1\}^s$ *as seguintes funções:*

$$g_{iw}(x_1, \ldots, x_k) = \begin{cases} 1 & \text{se } x_1, \ldots, x_k \text{ indexa a } j\text{-ésima linha de } A_i \text{ e } w_j = 1 \\ 0 & \text{caso contrário} \end{cases}$$

O "j" pode ser qualquer inteiro menor do que $|A_i|$*.*

$$h_{iw}(x_1, \ldots, x_{n-k}) = \begin{cases} 1 & \text{se a coluna de } A_i \text{ indexada por } x_1, \ldots, x_{n-k} \text{ for } w \\ 0 & \text{caso contrário} \end{cases}$$

Então podemos escrever $f(x_1, \ldots, x_n)$ *como*

$$\bigvee_{i=1}^{p} \bigvee_{w \in \{0,1\}^s} (g_{iw}(x_1, \ldots, x_k) \wedge h_{iw}(x_{k+1}, \ldots, x_n)) \tag{2.13}$$

Vamos nos convencer que f realmente pode ser escrita como 2.13. Particione $x \in \{0,1\}^n$ de forma que $x^1 = x_1 \ldots x_k$ e $x^2 = x_{k+1} \ldots x_n$, e suponha que x^1 indexe a j-ésima linha de A_i. Suponha que $f(x_1, \ldots, x_n) = 1$, o que implica na entrada (x^1, x^2) da tabela ser 1. Seja w a string que representa a coluna indexada por x^2 em A_i. Temos que $g_{iw}(x^1) = 1$ pois $(x^1, x^2) = w_j = 1$ e também $h_{iw}(x^2) = 1$ pela forma como escolhemos w, e portanto 2.13 é verdadeira também.

Indo na outra direção, se 2.13 é verdadeira então existe um w tal que $g_{iw}(x^1) = 1$ e $h_{iw}(x^2) = 1$, o que significa que uma string que representa a coluna indexada por x^2 tem um 1 na j-ésima linha de A_i, a linha indexada por x^1, e portanto a entrada (x^1, x^2) da tabela é 1 e consequentemente $f(x) = 1$.

Lema 2.60. *Para $n > 1$, podemos computar todos os monômios $x_1^{\alpha_1} \dots x_n^{\alpha_n}$ usando $\mathcal{O}(2^n)$ portas lógicas.*

Demonstração.

Provamos por indução que podemos computar todos monômios de n variáveis usando $2^{n+1} + n - 5$ portas lógicas.

Para computar $x_1^{\alpha_1} x_2^{\alpha_2}$, $(\alpha_1, \alpha_2) \in \{0,1\}^2$, precisamos usar apenas 4 portas $\wedge$ e 1 porta $\neg$ somando ao todo $5 = 2^{2+1} + 2 - 5$ portas lógicas.

Assumimos agora que para todo $2 < k < n$ é verdade que podemos computar todos monômios $x_1^{\alpha_1} \dots x_k^{\alpha_k}$ usando $2^{k+1} + k - 5$ portas lógicas. Então em particular, podemos calcular todos monômios de $n-1$ variáveis usando $2^n + (n-1) - 5$ portas lógicas. Para computar os monômios de n variáveis nós podemos usar os circuitos que computam os monômios de $n-1$ variáveis e ligamos x_n e $\overline{x_n}$ a cada uma das 2^{n-1} saídas dos monômios de $n-1$ variáveis adicionando 2^n portas $\wedge$ e uma porta $\neg$. Portanto ao todo nós usamos $(2^n + (n-1) - 5) + 2^n + 1 = 2^{n+1} + n - 5$ portas lógicas.

□

Agora mostramos como usamos a (k, s)-representação de Lupanov para alcançar um limite superior próximo do limite inferior de Shannon. A prova usada aqui pode ser encontrada em [FM05] e [Sav98]

Teorema 2.61. *Para todas as funções booleanas f, $f \in$ SIZE$((1 + o(1))\frac{2^n}{n})$.*

Demonstração. Vamos analizar o número de portas lógicas que precisamos para representar uma função booleana em sua (k, s)-representação de Lupanov com $k = \lceil 3 \log n \rceil$ e $s = \lceil n - 5 \log n \rceil$.

Primeiro nós computamos todos os monômios $x_1^{\alpha_1} \dots x_k^{\alpha_k}$ e $x_{k+1}^{\alpha_{k+1}} \dots x_n^{\alpha_n}$, o que nós já vimos em 2.60 que podemos fazer usando $\mathcal{O}(2^k + 2^{n-k})$ portas lógicas. Substituindo k por $\lceil 3 \log n \rceil$:

$$\mathcal{O}(2^{\lceil 3 \log n \rceil} + 2^{n - \lceil 3 \log n \rceil}) = \mathcal{O}(n^3 + \frac{2^n}{n^3}) = \mathcal{O}(\frac{2^n}{n^3})$$

Agora nós podemos computar cada g_{iw} e h_{iw} usando suas representações por $\vee$s de mintermos. Como todos os mintermos já foram previamente computados nós só precisamos adicionar portas $\vee$ para implementar essas funções.

O número de atribuições às variáveis $x_1, \dots, x_k$ que satisfazem g_{iw} é igual ao número de 1s em w então precisamos em média de $s/2$ portas $\vee$ para implementar cada g_{iw}. Então o número total de portas $\vee$ necessárias para implentar todos os g_{iw} é $sp2^{s-1}$. Como $p = \lceil \frac{2^k}{s} \rceil < (\frac{2^k}{s} + 1)$:

$$\begin{aligned} sp2^{s-1} &= s\lceil \frac{2^k}{s} \rceil 2^{s-1} \\ &< s(\frac{2^k}{s} + 1)2^{s-1} \\ &= 2^{k+s-1} + s2^{s-1} \end{aligned}$$

Substituindo $k = \lceil 3 \log n \rceil$ e $s = \lceil n - 5 \log n \rceil$ isso vira:

$$2^{\lceil 3 \log n \rceil + \lceil n - 5 \log n \rceil - 1} + (\lceil n - 5 \log n \rceil) 2^{\lceil n - 5 \log n \rceil - 1} < \frac{2^{n+1}}{n^2} + (n - 5 \log n + 1)\frac{2^n}{n^5} = \mathcal{O}(\frac{2^n}{n^2})$$

Cada atribuição de valores às variáveis $x_{k+1}, \dots, x_n$ satisfaz exatamente p funções h_{iw} e portanto o número de portas $\vee$ necessárias para implementar todas as h_{iw} é $p2^{n-k}$. Daí nós temos:

$$\begin{aligned} p2^{n-k} &= \lceil \frac{2^k}{s} \rceil 2^{n-k} \\ &< (\frac{2^k}{s} + 1)2^{n-k} \\ &= \frac{2^n}{s} + 2^{n-k} \end{aligned}$$

De novo substituindo os valores de k e s nós obtemos:

$$\frac{2^n}{\lceil n - 5\log n \rceil} + 2^{n-\lceil 3\log n \rceil} < \frac{2^n}{n - 5\log n - 1} + \frac{2^{n+1}}{n^3} = \mathcal{O}(\frac{2^n}{n})$$

E finalmente precisamos de $3p2^s = \mathcal{O}(\frac{2^{k+s}}{s}) = \mathcal{O}\big(\frac{2^n}{n^3}\big)$ portas $\vee$ e $\wedge$ que aparecem na fórmula 2.13.

Então o número total de portas lógicas necessárias para implementar a (k, s)-representação de Lupanov de f é:

$$\mathcal{O}(\frac{2^n}{n^3}) + \mathcal{O}(\frac{2^n}{n^2}) + \mathcal{O}(\frac{2^n}{n}) + \mathcal{O}(\frac{2^n}{n^3})$$

Ignorando os termos de menores ordem dentro dos $\mathcal{O}()$ nós temos que o número de portas lógicas é algo da ordem de $(1 + \frac{1}{n} + \frac{2}{n^2})\frac{2^n}{n} = (1 + o(1))\frac{2^n}{n}$.

□

Circuitos de profundidade logaritmica e P vs NC

Nesta subseção nós vamos conhecer duas classes de circuitos que contêm apenas circuitos de profundidade logaritmica. Nós também rapidamente fazemos alguns comentários a respeito de como essas classes de complexidade de circuitos se relacionam com computação paralela.

Nós dizemos que um problema é eficientemente paralelizável se ele pode ser computado em tempo polilogaritmico usando um número polinomial de processadores. Nós vimos na seção 2.5 que um problema é eficientemente computável se existe uma máquina de Turing que é capaz de decidir tal problema em tempo polinomial. Agora, o que se espera é que aumentar o número de "máquinas de Turing" nos permite decidir um dado problema mais rapidamente. O quão mais rapidamente? Saber se todos os problemas em P podem ser *altamente paralelizáveis* ainda é um problema em aberto pelo menos quase tão interessante quanto a questão P vs NP. A seguir nós vemos que a classe NC proposta por Pippenger - o primeiro nome dele é Nick, daí que veio o nome NC: *Nick's class* - que é equivalente à classe de problemas eficientemente paralelizáveis.

Definição 2.62. *(A classe NC)*

Uma linguage L é dita estar em NC^i se L é decidida por uma família de circuitos logspace-uniforme $\{C_n\}_{n\in\mathbb{N}}$ de tamanho polinomial tal que $DEPTH(C_n) = \mathcal{O}(\log^i n)$, para todo n.

A classe NC é $\bigcup_{i\geq 1} NC^i$.

Se $L \in NC$ então L é eficientemente paralelizável: se L é decidida por uma família de circuitos $\{C_n\}_{n\in\mathbb{N}}$ com profundidade $\mathcal{O}(\log^d n)$ então para entradas de tamanho n nós temos um único processador que computa a descrição de C_n e daí para cada porta lógica a de C_n nós temos um processador que irá computar $val(a)$ e enviar o resultado para o processador de cada porta lógica que depende de a. Nós precisamos de tempo

$\mathcal{O}(\log^d n)$ para computar a descrição de C_n e também para computar o valor de cada porta lógica. E pelo tamanho de C_n ser polinomial temos que o número de processadores também é polinomial.

Para ver que a outra direção também é verdade nós contruimos um circuito que nós vemos como um grid onde a porta lógica c_{ij} simula o i-ésimo processador no j-ésimo passo. Este circuito tem tamanho $\mathcal{O}(n^c \log^d n)$ e profundidade $\mathcal{O}(\log^d n)$.

Um exemplo de linguagem em NC é multiplicação de matrizes, que contém todas triplas $\langle M_1, M_2, M_1 M_2 \rangle$, e o problema de decidir a determinante de uma matrix que contém $\langle M, \mathrm{DET}(M) \rangle$. Ou seja, os problemas de multiplicar matrizes e computar a determinante de uma dada matriz são problemas eficientementes paralelizáveis.

A classe AC é definida como uma modificação de NC que permite fan-in arbitrário para suas portas lógicas.

Definição 2.63. *Uma linguagem L é dita estar em AC^i se L é decidida por uma família de circuitos logspace-uniforme $\{C_n\}_{n\in\mathbb{N}}$ de tamanho polinomial tal que $DEPTH(C_n) = \mathcal{O}(\log^i n)$, para todo n, e além disso o fan-in das portas lógicas são arbitrários.*

A classe AC é $\bigcup_{i\geq 1} AC^i$

Note que apesar do fan-in ser arbitrário, não faz sentido ter fan-in maior do que o número de variáveis de entrada e portanto podemos assumir que o fan-in máximo permitido pela classe AC é n quando n for o tamanho da entrada. Uma consequência disso é que podemos substituir uma porta lógica com fan-in $k \leq n$ por uma árvore de portas lógicas do mesmo tipo em que esta árvore tem profundidade $\log k \leq \log n$, o número adicional de portas lógicas é menor do que $2^{\log k} \leq 2^{\log n} = n$ e cada porta tem fan-in 2. O que isso significa é que podemos pegar um circuito C_n para uma linguagem em em AC^i e transformá-lo de forma que toda porta lógica no circuito resultante tenha fan-in 2 e a profundidade do circuito cresce por um fator logaritmico enquanto que o tamanho do circuito cresce por um fator linear. Então temos a seguinte relação entre NC e AC.

$$\mathsf{NC}^0 \subseteq \mathsf{AC}^0 \subseteq NC^1 \subseteq \mathsf{AC}^1 \subseteq NC^2 \subseteq \ldots,$$

o que nos dá $\mathsf{AC} = \mathsf{NC}$.

Mais pra frente a classe AC^0 será importante para nós pois vamos querer provar que certas linguagens não estão em AC^0 [3]. Nós vamos aproveitar esta seção para definirmos o que é um circuito AC^0 e também alguns termos que iremos usar extensivamente ao longo deste trabalho.

[3]Note que apesar de AC^0 ser uma classe que por sua definição parece ser bem fraca ela ainda está na fronteira de onde limitantes inferiores para problemas naturais foram obtidos. Por exemplo, ainda está em aberto se $\mathsf{NEXP} \subseteq AC^1$, muito menos é conhecido se SAT ou qualquer outra linguagem NP-completa não estão em AC^1

Capítulo 3

Computação relativizada e complexidade de circuitos

Neste capítulo nós vemos como circuitos e complexidade "relativizada" se relacionam.

3.1 Uma prova alternativa do teorema 2.45

Nós já vimos que existem oráculos A e B satisfazendo $\mathsf{P}^A = \mathsf{NP}^A$ e $\mathsf{P}^B \neq \mathsf{NP}^B$. Neste capítulo nós vemos uma outra prova do segundo resultado que mostra a idéia geral de como conectamos limites inferiores a resultados em complexidade relativizada. Basicamente, nós usamos a incapacidade de certos dispositivos computacionais (por exemplo, circuitos de baixa profundidade) de computar certas funções mais o fato que certas classes de complexidade podem ser expressas por estes mesmos dispositivos computacionais para, ironicamente, diagonalizar e provar separação por oráculo. Note que essa é a mesma idéia que usamos anteriormente para provar o teorema 2.45, porém se tentassemos generalizar aquele argumento para provar que $\mathsf{PH}^A \neq \mathsf{PSPACE}^A$ nós esbarrariamos em alguns problemas como por exemplo ter que simular cada máquina de Turing um número exponencial de vezes o que destroi a nossa hipótese que uma máquina de Turing de tempo polinomial só faz uma quantidade polinomial de consultas ao oráculo, e também podemos ver que trabalhar com modelos que apresentam uma estrutura combinatórica como circuitos Booleanas é muito menos problemático.

Nós iremos considerar a função Tribes_N que vimos na introdução. Uma das propriedades de Tribes_N é que ela é *evasiva*, o que significa que ter conhecimento apenas parcial dos bits da entrada não é suficiente para avaliar a função. Para formalizar isso, suponha que nós temos uma função $f : \{0,1\}^n \to \{0,1\}$ e queremos computar $f(x)$ para alguma entrada x. Nós podemos fazer isso somente consultando cada bit de x sem nos preocuparmos com a complexidade das computações intermediárias. Desta formas podemos definir diversas medidas de complexidade de f. A medida que nos interessa no momento é a complexidade de consulta determinística:

Definição 3.1. *A complexidade de consulta determinística de um algoritmo A é o maior número de consulta aos bits da entrada sobre todas as entradas $x \in \{0,1\}^n$ que A faz.*

Seja $f : \{0,1\}^n \to \{0,1\}$. A complexidade de consulta determinística de f, que denotamos por $D(f)$, é o mínimo da complexidade de consulta determinística de todos os algoritmos que computam f.

A partir de agora quando dissermos complexidade de consulta fica implícito que estamos falando da complexidade de consulta determinística.

Nós podemos representar as consultas feita pelo algoritmo através de uma árvore de decisão, onde em cada consulta o algoritmo ramifica para a esquerda ou para direita dependendo do valor do bit consultado. Desta forma $D(f)$ é a profundidade mínima entre todas as árvores de decisão que computam f.

Para ver que Tribes_N é evasiva bastar considerar o cenário em que nenhuma consulta revela o valor de uma das tribos (isto é, a consulta sempre retorna 1) até que o último bit da tribo seja consultado e acaba sendo 0. Assim mesmo que todas exceto uma tribo tenha todos seus bits revelados o valor desta última tribo vai ser incerto até que a última consulta seja feita. Isso pode ser traduzido como $D(\text{Tribes}_N) = N$. Também poderiamos ter usado o seguinte fato:

Fato 3.2. *Para todas funções $f : \{0,1\}^n \to \{0,1\}$, $deg(f) \leq D(f)$.*

E como vimos em **??**, $\deg(\text{Tribes}_n) = \text{n}$.

Agora, seja M uma máquina de Turing que pode fazer consultas a algum oráculo, e seja $x \in \{0,1\}^*$. O que M faz antes de tua primeira consulta é independente de qual oráculo ela tem acesso, e o que M faz entre a primeira e segunda consulta só depende da resposta à primeira consulta, e por aí vai. Isto sugere que a computação de M sobre a entrada x pode ser representada por um algoritmo de consulta que ramifica sobre as respostas às suas consultas ao oráculo.

Seja $\mathcal{X}$ um subconjunto finito de $\{0,1\}^*$ e $A \subseteq \{0,1\}^* \setminus \mathcal{X}$ um oráculo que inicialmente está definido apenas nas strings que não estão em $\mathcal{X}$. Sejam M e x os mesmos do parágrafo anterior, então o algoritmo de consulta $T^{\mathcal{X}}_{M^A,x}$ ramifica sobre consultas ao oráculo por strings em $\mathcal{X}$. Podemos ver a entrada de $T^{\mathcal{X}}_{M^A,x}$ como um vetor característica v onde $v_j = 1$ significa que a j-ésima string na ordem lexicográfica em $\mathcal{X}$ está em A. Desta forma, temos que $T^{\mathcal{X}}_{M^A,x}(v) = 1 \iff M^{A \cup v}(x) = 1$, permitindo um pequeno abuso de notação. Em particular, se M roda em tempo polinomial e $\mathcal{X} = \{0,1\}^{|x|}$ então $T^{\mathcal{X}}_{M^A,x}$ tem complexidade de consulta polilogarítmica se $|x|$ for suficientemente grande.

Então, para cada oráculo $A \subseteq \{0,1\}^*$, considere a seguinte linguagem.

$$L_A = \{1^w | \text{Tribes}_{w,s}(A) = 1\},$$

em que $s \approx 2^w \ln(2)$. Se $N = ws$, o número de variáveis de entrada de $\text{Tribes}_{w,s}$, então $\text{Tribes}_{w,s}(A)$ siginifica a função $\text{Tribes}_{w,s}$ com o vetor característica de $A^{=\log N}$ em seu argumento. $L_A \in \mathsf{NP}^A$ para todos $A \subseteq \{0,1\}^*$ pois precisamos apenas verificar que para algum i múltiplo de w, $x_{i+1}, x_{i+2}, \ldots, x_{i+w} \in A$, em que aqui x_i é a i-ésima string de tamanho $\log N$ na ordem lexicográfica. Como $\log N \approx w + \log w$ também temos que é possível em tempo polinomial escrever cada uma dessas strings na fita de oráculo. Agora podemos provar o teorema 2.45 novamente:

Proposição 3.3. *Existe um oráculo $A \subseteq \{0,1\}^*$ tal que $L_A \notin P^A$.*

Demonstração. Seja $\{M_i\}_{i \geq 1}$ uma enumeração de todas as máquinas de Turing de tempo polinomial, e $\{p_i\}_{i \geq 1}$ o tempo de execução das respectivas máquinas de Turing.

Seja $w_1, w_2, \ldots$ uma sequência de números naturais em que $p_1(w_1) < 2^{w_1}$ e para cada $i > 1$ temos que $w_i > p_{i-1}(w_{i-1})$ e $p_i(w_i) < 2^{w_i}$.

Assim como fizemos na prova original que aparece na seção 2.5 nós iremos construir o oráculo A em estágios. Nós descrevemos o i-ésimo estágio para algum $i \geq 1$ arbitrário. Inicialmente fazemos $A(0) = \emptyset$ e iremos assumir que no fim do $(i-1)$-ésimo a construção descrita abaixo já obteve o oráculo $A(i-1)$ definido sobre strings de tamanho no máximo $p_{i-1}(w_{i-1})$

Estágio i:

Seja $D(i) \subseteq \bigcup_{k \in I_i} \{0,1\}^k$, em que $I_i = \{p_{i-1}(w_{i-1}) + 1, p_{i-1}(w_{i-1}) + 2, \ldots, p_i(w^i)\} \setminus \{0,1\}^{\log N_i}$ e $N_i \approx$

$w_i + \log w_i$ [1]. Seja T_i a árvore de decisão da computação de M_i sobre a entrada 1^{w_i} com acesso ao oráculo $A(i-1) \cup D(i)$ em que os nodos da árvore ramificam com consutas à strings em $\{0,1\}^{\log N_i}$. Como T_i tem profundidade polilogaritmica e Tribes_{N_i} é evasiva, segue que existe um oráculo $A' \subseteq \{0,1\}^{\log N_i}$ tal que $T_i(A') \neq \text{Tribes}_{N_i}(A')$. Nós então fazemos $A(i) = A(i-1) \cup A' \cup D(i)$.

Então agora argumentamos que $A = \bigcup_{i \geq 1} A(i)$ satisfaz $\mathsf{NP}^A \not\subseteq P^A$. Nós iremos assumir que para algum $i \geq 1$ a máquina de Turing M_i decide L_A com acesso à A em tempo polinomial. Nós fazemos $A' = A \cap \{0,1\}^{\log N_i}$. Este A' apareceu no i-ésimo estágio e é tal que $A(i-1) = A(i) \setminus (A' \cup D(i))$. Nós temos o seguinte.

- $1^{w_i} \in L_A \Rightarrow T_i(A') = 1 \Rightarrow \text{Tribes}_{N_i}(A) = 0 \Rightarrow 1^{w_i} \notin L_A$.
- $1^{w_i} \notin L_A \Rightarrow T_i(A') = 0 \Rightarrow \text{Tribes}_{N_i}(A) = 1 \Rightarrow 1^{w^i} \in L_A$.

Ambos os casos são contradições e portanto tal máquina M_i não pode existir e $\mathsf{NP}^A \not\subseteq P^A$.

□

Na prova que acabamos de ver nós usamos dois fatos a respeito da função Tribes_N: 1) ela é evasiva (na verdade só precisamos do fato que Tribes_N não pode ser computada por um algoritmo que faz $\Omega(N^c)$ consultas, para qualquer $c < 1$) e 2) nós podemos verificar que $\text{Tribes}_N = 1$ olhando para um número polinomial das variáveis de entrada de Tribes_N. Portanto poderiamos substituir Tribes_N por qualquer função que satisfaça (1) e (2), como por exemplo a função Or_n, a disjunção de n variáveis. Porém, a função Or_n não é balanceada, o que significa que com probabilidade muito maior do que $1/2$ temos que $Or_n(x) = 1$. O problema é que nós iremos precisar de uma função balanceada para provar o resultado da próxima seção.

3.2 P ≠ NP para oráculos aleatórios

Um dos objetivos de estudar complexidade relativizada é enteder a relação entres classes de complexidades quando não conseguimos dizer nada de muito útil no mundo não-relativizado. Porém, como já vimos pelo teorema 2.45, provar que $\mathsf{P} \neq \mathsf{NP}$ para algum oráculo não é nenhuma evidência que $\mathsf{P} \neq \mathsf{NP}$ no mundo não-relativizado, até porque a construção do oráculo A em ambas provas que vimos é só uma especialização do método da diagonalização que por sua vez é basicamente só uma forma de "trapacear o sistema" construido uma asserção que codifica a sua própria falsidade. Então, com o objetivo de conseguir algum tipo de evidência que $\mathsf{P} \neq \mathsf{NP}$ poderiamos nos perguntar se é o caso que P e NP são diferentes no mundo relativizado típico. E de fato, o próximo teorema é o assunto desta subseção:

Teorema 3.4. $Pr_A[\mathsf{P}^A \neq \mathsf{NP}^A] = 1$.

A idéia que teoremas como 3.4 é evidência que o mesmo resultado que foi provado acontecer quase sempre para oráculos aleatórios também há de ser verdade no mundo não relativizado é conhecido como a hipótese do oráculo aleatório que iremos discutir na seção 3.5. Naquela mesma seção iremos ver que a hipótese é falsa.

Antes de ver a prova do teorema 3.4 nós vamos rapidamente discutir o que queremos dizer por $\mathsf{P}^A \neq \mathsf{NP}^A$ com probabilidade 1.

A lei zero-um de Kolmogorov

Um oráculo aleátorio é gerado incluindo cada $x \in \{0,1\}^n$ independentemente com probabilidade $1/2$.

[1] Neste ponto nós podemos assumir que $p_i = \Omega(n \log n)$ e que $p_i(w_i) > w_i + \log w_i$, pois o teorema da hierarquia de tempo determinístico que vimos em 2.43 diz que se nenhuma máquina de Turing que roda em tempo $\Omega(n \log n)$ pode decidir uma linguagem qualquer então certamente nenhuma máquina que roda em tempo $\mathcal{O}(n)$ pode decidir tal linguagem.

Prova do teorema 3.4

A idéia principal do teorema 3.4 é que a função $\text{Tribes}_{w,s} = \text{Tribes}_N$ não pode nem mesmo ser aproximada por algoritmos de consulta com complexidade de consulta polilogarítmica.

Proposição 3.5. *Seja A qualquer algoritmo de consulta com complexidade de consulta $o(\frac{n}{logn})$, então:*

$$\Pr_{x\sim\{0,1\}^n}[A(x) = \mathit{Tribes}_N(x)] \leq 0,567$$

Demonstração. É suficiente provar que qualquer algoritmo A que só faz consultas às primeiras $\frac{1}{10}s$ tribos não pode aproximar a função Tribes_N. Podemos notar que neste caso o melhor que podemos fazer é fazer A computar a função $\text{Tribes}_{\frac{1}{10}s,w}$. Ou seja, se g é a função computada por A, então

$$g(x) = 1 \iff \text{pelo menos uma das primeiras } \frac{1}{10}s \text{ tribos é unanimamente 1.}$$

Daí temos que

$$\Pr_{x\sim\{0,1\}^N}[\text{Tribes}_N(x) \neq g(x)] = \mathsf{E}_{x\sim\{0,1\}^N}[(\text{Tribes}_N(x) - g(x))^2].$$

Mas como $\text{Tribes}_N(x) \geq g(x)$, para todos $x \in \{0,1\}^N$, temos que

$$\begin{aligned}\Pr_{x\sim\{0,1\}^N}[\text{Tribes}_N(x) \neq g(x)] &= \mathsf{E}_{x\sim\{0,1\}^N}[\text{Tribes}_N(x) - g(x)] \\ &= \mathsf{E}_{x\sim\{0,1\}^X}[\text{Tribes}_N(x)] - \mathsf{E}_{x\sim\{0,1\}^n}[g(x)] \\ &= 1 - (1-2^{-w})^s - 1 + (1-2^{-w})^{\frac{1}{10}s} \\ &= (1-2^{-w})^{\frac{1}{10}s} - (1-2^{-w})^s\end{aligned}$$

Com w tendendo ao infinito isso é igual a $2^{-\frac{1}{10}} - 1/2 \approx 0,433$, e portanto

$$\Pr_{x\sim\{0,1\}^N}[\text{Tribes}_N(x) = A(x)] \leq 1 - 0,433 = 0,567.$$

□

Então podemos provar o teorema 3.4.

Demonstração. (Prova do teorema 3.4)

Pela proposição 3.5 nós temos que para uma fração significativa das entradas, $T_i(\boldsymbol{A'}) \neq \mathit{Tribes}_N(\boldsymbol{A'})$ e portanto o conjunto $\{A \subseteq \{0,1\}^* | \mathsf{P}^A \neq \mathsf{NP}^A\}$ tem medida positiva.Usando a lei zero-um de Kolmogorov podemos concluir que $\Pr[\mathsf{P}^{\boldsymbol{A}} \neq \mathsf{NP}^{\boldsymbol{A}}] = 1$.

□

3.3 PH vs PSPACE

Agora nós generalizamos a idéia que usamos para separar P e NP nas seções anteriores para podermos separar PSPACE e a hierarquia polinomial. Da mesma forma que usamos árvores de decisão para representar P nós iremos usar circuitos AC^0 para representar PH.

Fixe $k \geq 1$. Seja $L \subseteq \{0,1\}^*$ uma linguagem em Σ_k^p, o que significa que existe uma máquina de Turing M de tempo polinomial tal que para todo $x \in \{0,1\}^*$, $x \in L \iff \exists y_1 \forall y_2 \ldots Q_k y_k M(x, y_1, y_2, \ldots, y_k) = 1$. Fixando $y_1, y_2, \ldots, y_k$, podemos representar a computação de M sobre $x, y_1, y_2, \ldots, y_k$ por uma árvore de decisão da forma que vimos anteriormente. Então o circuito AC^0 $C^{\mathcal{X}}_{M,x}$ que usamos para representar L tem profundidade $k+1$ onde cada quantificador $\exists$ é representado por uma camada de portas $\vee$ e $\forall$ é representado por uma camada de portas $\wedge$, a profundidade é $k+1$ porque no nível mais baixo nós temos árvores de decisão que podem ser representadas por fórmulas FNC ou FND. Se $N = 2^n$ é o número de variáveis de entrada do circuito $C^{\mathcal{X}}_{M,x}$, também vemos que o fan-in das portas no primeiro nível é $\mathsf{polylog}(N)$ e o fan-in nas demais portas é $2^{p(n)}$. O tamanho do circuito é $N^{\mathsf{polylog}(N)}$.

Assim como as árvores de decisão que vimos nas seções 3.1 e 3.2, o circuito $C^{\mathcal{X}}_{M,x}$ satisfaz $C^{\mathcal{X}}_{M,x}(A) = 1 \iff M^{\mathcal{X} \cup A}(x) = 1$, onde nós vemos A como sendo o vetor caratecterísica de um oráculo A.

O objetivo desta seção é provar o seguinte teorema.

Teorema 3.6. *Existe $A \subseteq \{0,1\}^*$ tal que $\mathsf{PSPACE}^A \neq \mathsf{PH}^A = \bigcup_{k \geq 1} \Sigma_k^{p,A}$.*

Nós também temos que $\mathsf{PH}^{\mathsf{TBQF}} = \mathsf{PSPACE}^{\mathsf{TBQF}}$, e portanto o teorema 3.6 também nós diz que PH vs PSPACE, assim como P vs NP, necessariamente precisa de um argumento que não relativiza. Nós provamos 3.3 a partir de um limitante inferior para a função Tribes_N. Para provar o teorema 3.6 nós iremos usar um limitante inferior para a função paridade de n variáveis.

Definição 3.7. *(Paridade)*

Para $n \geq 1$ a função paridade de n variáveis, que denotaremos por Parity_n, é 1 se e se somente se $\sum_{i=1}^n x_1 \pmod 2 = 1$.

Consideremos para cada oráculo $A \subseteq \{0,1\}^*$ a seguinte linguagem:

$$L(A) = \{1^n | A(x) = 1 \text{ para um número ímpar de } x \in \{0,1\}^n\}.$$

E temos que $L(A) \in \mathsf{PSPACE}^A$ já que só precisamos de espaço para escrever cada string de n bits na fita de oráculo mais uma célula para guardar a paridade de $|\{x \in \{0,1\}^n | A(x) = 1)\}|$. Uma outra forma de descrever L é dado uma string unária 1^n e oráculo A, então $1^n \in L(A) \iff \text{Parity}_{2^n}(A^{=n}) = 1$ em que $A^{=n} = A \cap \{0,1\}^n$. Nós iremos usar esta caracterização de $L(A)$ mais o teorema 3.8 que é enunciado logo abaixo para provar o teorema 3.6.

Teorema 3.8. *Seja $d > 0$ um inteiro. Para n suficientemente grande temos que qualquer circuito de profundidade d com fan-in $\mathsf{polylog}(n)$ no teu primeiro nível e tamanho $< 2^{\mathcal{O}\left(n^{\frac{1}{d-1}}\right)}$ não pode computar a função Parity_n corretamente em todas as entradas.*

Nós iremos provar o teorema 3.8 no próximo capítulo (4) quando formos ver restrições aleatórias e o lema da troca de Håstad. Na verdade, iremos ganhar de grátis o seguinte teorema.

Teorema 3.9. *Seja $d > 0$ um inteiro. Então, para n suficientemente grande, qualquer circuito com profundidade d, fan-in do primeiro nível $\mathsf{polylog}(n)$ que computa Parity_n corretamente numa fração maior do que $1/2 + \Omega(n^{-d})$ das entradas deve ter tamanho maior do que $2^{\Theta\left(n^{\frac{1}{d-1}}\right)}$.*

E portanto, usando a lei zero-um de Kolmogorv e um argumento semelhante ao que usamos para provar 3.4, nós temos o seguinte.

Teorema 3.10. $\Pr_A[\mathsf{PSPACE}^A \neq \mathsf{PH}^A] = 1$.

Demonstração. (Prova do Teorema 3.6)

Nós usamos um argumento parecido com o da prova do teorema 3.3.

Seja $\{M_i\}_{i\geq 1}$ uma enumeração de todas as máquinas de Turing de tempo polinomial e $\{p_i\}_{i\geq 1}$ o tempo de execução dessas máquinas de Turing. Fixe algum $k \geq 1$ e provaremos que existe $A \subseteq \{0,1\}^*$ tal que $\Sigma_k^{p,A} \neq \mathsf{PSPACE}^A$. Por estarmos escolhendo um k arbitrário, após provarmos que $\Sigma_k^{p,A} \neq \mathsf{PSPACE}^A$ teremos também provado que $\mathsf{PH}^A \neq \mathsf{PSPACE}^A$.

Seja n_0 a constante tal que para todo $n \geq n_0$ o resultado do teorema 3.8 para circuitos de profundidade $k+1$ vale – ou seja, para todo $n \geq n_0$ e todo circuito C de profundidade $k+1$ e tamanho subexponencial existe um $x \in \{0,1\}^n$ tal que $C(x) \neq \text{Parity}_n(x)$. Nós definimos uma sequência $n_0, n_1, n_2, \ldots$ de números naturais tais que para todo $i \geq 1$ é verdade que $n_i > p_{i-1}(n_{i-1})$ e $p_i(n_i) < 2^{n_i}$.

Nós construimos A em estágios e descreveremos o i-ésimo estágio para algum $i \geq 1$ arbitrário. Seja $A(0) = \emptyset$.

Estágio i:

Seja $D(i) \subseteq \bigcup_{k\in I_i}\{0,1\}^k$ algum oráculo arbitrário onde $I_i = \{p_{i-1}(n_{i-1})+1, p_{i-1}(n_{i-1})+2, \ldots, p_i(n_i)\} \setminus \{0,1\}^{n_i}$. Seja C_i o circuito da computação de M_i sobre a entrada 1^{n_i} e com acesso ao oráculo $A(i-1) \cup D(i)$ e que faz consultas à strings em $\{0,1\}^{n_i}$. Pelo teorema 3.8 deve haver um oráculo $A' \subseteq \{0,1\}^{n_i}$ tal que $C_i(A') \neq \text{Parity}_{2^{n_i}}(A')$. Então fazemos $A(i) = A(i-1) \cup A' \cup D(i)$.

Argumentaremos que $A = \bigcup_{i=1}^{\infty} A(i)$ satisfaz $L(A) \notin \Sigma_k^{p,A}$. Suponha que uma máquina de Turing de tempo polinomial M_i decida $L(A)$ com k quantificadores alternantes e com acesso a A. Seja A' o oráculo que apareceu no i-ésimo estágio que é tal que $A(i-1) = A(i) \setminus (A' \cup D(i))$. Note que também é verdade que $A \cap \{0,1\}^{n_i} = A'$. Então nós temos o seguinte.

- $1^{n_i} \in L(A) \Rightarrow C_i(A') = 1 \Rightarrow \text{Parity}_{2^{n_i}}(A') = 0 \Rightarrow 1^{n_i} \notin L(A)$.
- $1^{n_i} \notin L(A) \Rightarrow C_i(A') = 0 \Rightarrow \text{Parity}_{2^{n_i}}(A') = 1 \Rightarrow 1^{n_i} \in L(A)$.

Ambos os casos são contradições e portanto não existe tal máquina de Turing de tempo polinomial que decida $L(A)$ com k quantificadores alternantes. Como $L(A) \in \mathsf{PSPACE}^A$ podemos concluir que $\Sigma_k^{p,A} \neq \mathsf{PSPACE}^A$.

□

3.4 Separando a hierarquia polinomial

Como vimos na seção anterior, podemos representar Σ_k^p por circuitos AC^0 com profundidade $k+1$. Portanto, se queremos provar que existe $A \subseteq \{0,1\}^*$ tal que $\Sigma_k^{p,A} \not\subseteq \Sigma_{k-1}^{p.A}$ nós temos que demonstrar a existência de uma "hierarquia de profundidade". O que queremos dizer é que deve existir para cada $k > 1$ uma função f_k tal que existe um circuito de tamanho polinomial e profundidade $k+1$ que computa f_k mas que qualquer circuito com profundidade k que computa f_k tem tamanho exponencial. Note que pelo teorema 3.8 a função paridade não pode ser computada por circuitos AC^0 de tamanho polinomial e profundidade k para *todas* as constantes $k \geq 1$ e portanto temos que provar limites inferior para funções diferentes da função paridade. Para este fim define-se as funções de Sipser:

Definição 3.11. *(As funções de Sipser)*

Para $d \geq 2$ a função de Sipser $f^{m,d}$ é uma fórmula monotônica e read-once [2] *onde o nível mais baixo tem fan-in m, as portas lógicas nos níveis 2 até d - 1 têm fan-in $w = 2^m m \ln(2)$ e a porta lógica no nível mais alto tem fan-in $w_d \approx 2^m ln(2)$. Ou seja, podemos escrever $f^{m,d}$ como*

$$\bigvee_{i_d=1}^{w_d} \bigwedge_{i_{d-1}=1}^{w} \cdots \bigvee_{i_2=1}^{w} \bigwedge_{i_1=1}^{m} x_{i_1,i_2,\ldots,i_d}, \text{ se } d \text{ é par.} \tag{3.1}$$

e

$$\bigwedge_{i_d=1}^{w_d} \bigvee_{i_{d-1}=1}^{w} \cdots \bigvee_{i_2=1}^{w} \bigwedge_{i_1=1}^{m} x_{i_1,i_2,\ldots,i_d}, \text{ se } d \text{ é ímpar.} \tag{3.2}$$

Segue direto da definição que o número n de variáveis de entrada da função $f^{m,d}$ é simplesmente o produto dos fan-ins de cada nível.

$$n = mw^{d-2}w_d = \Theta(w^{d-1}).$$

E portanto também temos que $w = \Theta(n^{\frac{1}{d-1}})$.

Segue também pela definição de $f^{m,d}$ exposta em 3.1 e 3.2 que ela pode ser computada por um circuito de profundidade d e tamanho

$$S = 1 + \frac{n}{m}\sum_{i=0}^{d-2} n^i = 1 + \frac{n}{m}\Big(\frac{w^{d-1}-1}{w-1}\Big) = \widetilde{\Theta}(n^{2-\frac{1}{d-1}}),$$

Agora o resultado que nós precisamos para separar cada nível da hierarquia polinomial relativa à um oráculo segue do seguinte resultado, que também foi provado pela primeira vez por Håstad em sua tese de doutorado.

Teorema 3.12. *Seja $d > 2$ e m suficientemente grande, qualquer circuito de tamanho no máximo $2^{w^{1/5}}$ e profundidade $d-1$ não computa a função $f^{m,d}$ corretamente em todas as entradas.*

Como $w = \Theta(n^{\frac{1}{d-1}})$, o teorema 3.12 nos dá um limitante inferior de $2^{n^{\Omega\left(\frac{1}{d-1}\right)}}$ para o tamanho de qualquer circuito de profundidade $d-1$ que computa a função $f^{m,d}$. De novo, deixaremos a prova do teorema 3.12 para o próximo capítulo, por enquanto só estamos preocupados na seguinte aplicação deste teorema.

Teorema 3.13. *Existe um oráculo $A \subseteq \{0,1\}^*$ tal que para todo $k \geq 2$, $\Sigma_k^{p,A} \neq \Sigma_{k-1}^{p,A}$.*

Ou seja, para este oráculo A, $\mathsf{PH}^A \neq \Sigma_k^{p,A}$, para todos $k \geq 1$. Logo, em particular, $\mathsf{NP}^A \neq \mathsf{PH}^A \Rightarrow \mathsf{P}^A \neq \mathsf{NP}^A$, então podemos ver a prova do teorema 3.13 como a terceira prova do teorema de Baker-Gill-Solovay em 2.45 e 3.3 que iremos ver, mas desta vez também estaremos provando algo bem mais forte usando uma estratégia parecida. Nós na verdade iremos provar que existe um oráculo A tal que para todo $k \geq 2$ é verdade que $\Pi_k^{p,A} \neq \Pi_{k-1}^{p,A}$, mas como observamos ainda na seção 2.5, é verdade que $\bigcup_{k\geq 1} \Sigma_k^{p,A} = \bigcup_{k=1} \Pi_k^{p,A} = \mathsf{PH}^A$.

Para cada $k \geq 1$ nós definimos a linguagem $L_k(A) = \{1^m | f^{m,k+1}(A) = 1\}$, em que neste caso as variáveis de entrada de $f^{m,k+1}$ são $A(x)$ para cada string de x de uma tamanho $|x| = \log n$ fixo em que $n \approx (d-1)(m + \log m)$. Nós podemos notar que $L_k(A) \in \Pi_k^{p,A}$ pois podemos usar k quantificadores alternantes para simular as diferentes camadas do circuito e depois disso tudo que precisamos fazer é uma quantidade linear de consultas à A para achar o valor de uma porta $\wedge$ no nível mais baixo. Para provar que existe um oráculo A tal que $\Pi_k^{p,A} \neq \Pi_{k-1}^{p,A}$ nós iremos provar a asserção mais forte que diz que existe um oráculo A tal que $L_k(A) \notin \Pi_{k-1}^{p,A}$.

[2] Nós dizemos que um circuito ou fórmula é *read-once* se cada variável de entrada só alimenta uma única porta lógica.

Demonstração. (Prova do Teorema 3.13)

Nós agora iremos enumerar predicados PH que são predicados P da forma

$$P(x) \iff \forall z_1 \exists z_2 \forall \ldots Q_k z_k M(x, z_1, z_2, \ldots, z_k),$$

em que M é uma máquina de Turing de tempo polinomial. Seja $\{P_i\}_{i\geq 1}$ uma enumeração dos predicados PH e $\{p_i\}_{i\geq 1}$ é o tempo de execução de cada máquina de Turing na definição dos predicados PH.

Nós então consideramos uma sequência $m_0, m_1, m_2, \ldots$ de números naturais em que m_0 é grande o suficiente para satisfazer o resultado do teorema 3.12 e para todo $i \geq 1$ temos que $m_i > p_{i-1}(m_{i-1})$ e $p_i(m_i) < 2^{m_i}$. Para cada m_i nós teremos um outro número natural n_i que seria o número de variáveis de entrada da função $f^{m_i,k+1}$ em que k é tal que P_i é um predicado PH com $k-1$ quantificadores alternantes.

Como fizemos em todas as provas de resultados deste tipo, nós iremos construir o oráculo A em estágios e descreveremos o i-ésimo estágio, para algum $i \geq 1$ arbitrário. Inicialmente nós fazemos $A(0) = \emptyset$.

Estágio i:

Seja $k \geq 2$ tal que P_i é um predicado PH com $k-1$ quantificadores alternantes. Seja $D(i) \subseteq \bigcup_{l\in I_i}\{0,1\}^l$ um oráculo arbitrário em que $I_i = \{p_{i-1}(m_{i-1})+1, p_{i-1}(m_{i-1})+2, \ldots, p_i(m_i)\} \setminus \{0,1\}^{\log n_i}$ [3]. Seja C_i o circuito da computação de P_i sobre a entrada 1^{m_i} com $k-1$ quantificadores alternantes e acesso ao oráculo $A(i-1)\cup D(i)$. Ou seja, C_i é um circuito de profundidade k com tamanho quasipolinomial [4] e fan-in polilogaritmico no seu nível mais baixo e

$$C_i(A') = 1 \iff \forall z_1 \exists z_2 \forall \ldots Q_{k-1} z_{k-1} M_i^{A(i-1)\cup D(i)\cup A'}(1^{m_i}, z_1, z_2, \ldots, z_{k-1}) = 1.$$

Então, pelo teorema 3.12 deve haver um oráculo $A' \subseteq \{0,1\}^{\log n_i}$ tal que $C_i(A') \neq f^{m,k+1}(A')$. Nós então fazemos $A(i) = A' \cup D(i)$.

Agora suponha que existe um $k \geq 1$ e um predicado PH P_i com k quantificadores alternantes tal que $P_i(x) \iff x \in L_k(A)$. Nós então consideramos o oráculo $A' = A \cap \{0,1\}^{\log n_i}$. Nós temos o seguinte

- $1^{m_i} \in L_k(A) \Rightarrow C_i(A') = 1 \Rightarrow f^{m_i,k+1}(A') = 0 \Rightarrow 1^{m_i} \notin L_k(A)$.
- $1^{m_i} \notin L_k(A) \Rightarrow C_i(A') = 0 \Rightarrow f^{m_i,k+1}(A') = 1 \Rightarrow 1^{m_1} \in L_k(A)$.

Ambos os casos são contradições.

□

Separando a hierarquia polinomial com oráculos aleatórios

Recentemente, Rossman, Servedio e Tan (referência) provaram uma versão do teorema 3.12 para o caso médio. Ou seja, eles mostraram que as funções de Sipser da forma que estamos definindo elas neste texto nos dão uma hierarquia de circuitos de profundidade constante até mesmo no caso médio em que nós permitimos que circuitos de profundidade $d-1$ apenas aproximem circuitos de tamanho polinomial e profundidade d. Ou seja, o teorema 3.12 nos diz que circuitos de tamanho subexponencial e profundidade $d-1$ não são capazes de computar $f^{m,d}$ corretamente em todas as entradas. O teorema que iremos enunciar logo em seguida vai além e diz que circuitos de tamanho subexponencial e profundidade $d-1$ nem sequer podem aproximar $f^{m,d}$.

[3] Agora temos que $\log n_i \approx (k-1)(m_i + \log m_i)$ mas podemos de novo assumir que todo polinômio p_i é $\Omega(n \log n)$ sem perda de generalidade.

[4] Quasipolinomial significa funções da forma $n^{\log^c n}$ para $c > 0$.

Teorema 3.14. *Seja $d > 2$ e m suficientemente grande, então qualquer circuito de tamanho $S \leq 2^{w^{1/5}}$ e profundidade $d-1$ falha em computar a função $f^{m,d}$ corretamente numa fração maior do que $1/2 + n^{-\Omega(1/d)}$ das entradas.*

Teorema 3.15. $\Pr_{A \sim \{0,1\}^*}[\mathsf{NP}^A \neq \Sigma_2^{p,A}] = 1$.

E o teorema 3.15 segue do seguinte limite inferior para uma fórmula FNC que aproxima uma variação da função Tribes.

Definição 3.16. *(Dual da função Tribes)*

O dual da função $Tribes_{w,s}$ é a função $Tribes^{\dagger}_{w,s}$ definida como

$$Tribes^{\dagger}_{w,s}(x_{1,1}, x_{1,2}, \ldots, x_{s,w}) = \overline{\Big(\bigvee_{i=1}^{s} \bigwedge_{j=1}^{w} \overline{x}_{i,j} \Big)} = \overline{Tribes_{w,s}(\overline{x}_{1,1}, \overline{x}_{1,2}, \ldots, \overline{x}_{s,w})}.$$

E então provaremos o seguinte teorema.

Teorema 3.17. *(O'Donnell, Wimmer)*

Seja $F_m = Tribes_{b,2^b} \vee Tribes^{\dagger}_{b,2^b}$, então qualquer circuito FNC (ou FND) C que 0,04-aproxima f deve ter tamanho $s = 2^{\Omega(n/\log n)}$.

Como f pode ser computada por um circuito de tamanho polinomial e profundidade 3, o teorema 3.15 segue diretamente de 3.17.

Para provar o teorema 3.17 nós precisaremos de algumas observações. Caso quisermos computar a função $\text{Tribes}_{b,2^b}$ nós precisamos apenas saber o valor de cada tribo. Ou seja, sejam $\{y_i\}_{i=[2^b]}$ variáveis, nós podemos "projetar" as variáveis $\{x_{i,j}\}_{j \in [b]}$ para a variável y_i de forma que $y_i = \bigwedge_{j=1}^{b} x_{i,j}$, o que denotaremos por $(y|x)$, e então podemos ver $\text{Tribes}_{b,2^b}$ como a disjunção das variáveis y_i. Considere então uma fórmula FNC F que ε-aproxima a função $\text{Tribes}_{b,2^b}$ e que tenha tamanho s e cada cláusula tenha largura no máximo w, e considere também uma cláusula C qualquer de F e troque cada aparição da variável $x_{i,j}$ em C por $(x_{i,j} \vee y_i)$ e cada aparição de $\overline{x}_{i,j}$ por $\overline{(x_{i,j} \vee y_i)}$ e chame a disjunção resultante de C'. Podemos argumentar que C' é equivalente à C pois $(x_{i,j} \vee y_i)$ é sempre igual a $x_{i,j}$. Então, se F_1 for a fórmula FNC construida a partir de F da forma que acabamos de descrever (a conjunção das novas cláusulas C') temos que F_1 e F são equivalente, e daí segue que

$$\Pr_{x,(y|x)}[F_1(x,y) \neq \text{Tribes}_{b,2^b}(x)] < \varepsilon$$

Mas ao invés de construir y a partir de x podemos também construir x a partir de y sem mudar a distribuição de pares (x,y) da seguinte maneira:

1. Para cada $i \in [2^b]$, faça $y_i = 0$ com probabilidade $1 - 2^{-b}$ e $y_i = 1$ com probabilidade 2^{-b}.
2. Para cada $i \in [2^b]$ e $j \in [b]$, faça $x_{i,j} = 1$ se $y_i = 1$, e $(x_{i,1}, \ldots, x_{i,b})$ tirada da distribuição uniforme sobre $\{0,1\}^b \setminus (1,1,\ldots,1)$ se $y_i = 0$.

Denotamos a string x construida a partir de y por $(x|y)$, e então:

$$\mathsf{E}_y \Big[\Pr_{(x|y)}[F_1(x,y) \neq \text{Tribes}_{b,2^b}(x)] \Big] < \varepsilon$$

Agora, note que dado que $y_i = 1$ os valores de $x_{i,j}$ são meio que redudantes, então podemos trocar as strings $(x_{i,1}, \ldots, x_{i,b})$ por uma string z_i tirada da distribuição uniforme sobre $\{0,1\}^b \setminus (1,1,\ldots,1)$, independente de y. Seja $g(y) = \bigvee_{i=1}^{2^b} y_i$ que é equivalente à $\text{Tribes}_{b,2^b}(x)$. Então, temos o seguinte:

$$\mathsf{E}_y\Big[\Pr_z[F_1(z,y) \neq g(y)]\Big] < \varepsilon$$

e por z e y serem independentes:

$$\mathsf{E}_z\Big[\Pr_y[F_1(z,y) \neq g(y)]\Big] < \varepsilon$$

E portanto deve existir um $z^* \in \{0,1\}^b$ tal que

$$\Pr_y[F_1(z^*,y) \neq g(y)] < \varepsilon.$$

E definimos a fórmula FNC F' como $F'(y) = F_1(z^*, y)$. O que podemos tirar de tudo isso é que aproximar $\text{Tribes}_{b,2^b}$ por fórmulas FNC é pelo menos tão difícil quanto aproximar a função g sobre as variáveis $\{y_i\}_{i\in[2^b]}$. Agora nós podemos provar um limite inferior para a largura de um circuito FNC que aproxima a função $\text{Tribes}_{b,2^b}$.

Teorema 3.18. *Seja $m = b2^b$ o número de variáveis de entrada da função $\text{Tribes}_{b,2^b}$ e seja C um circuito FNC que 0,2-aproxima esta função com largura w. Então devemos ter $w \geq (1/3)2^b = \Omega(m/\log m)$.*

Demonstração. Vamos considerar o circuito C' sobre as variáveis $\{y_i\}_{i\in[2^b]}$ análogo à fórmula F' que acabamos de descrever. C' 0,2-aproxima a função $g(y) = \bigvee_{i=1}^{2^b} y_i$ e tem largura $w' \leq w$ e portanto provar que $w' \geq (1/3)2^b$ é suficiente para provar o teorema.

Vamos primeiro assumir que todas as cláusulas de C' tenha uma variável negada. Neste caso a entrada $(0,0,\ldots,0)$ faz C' avaliar para 1 enquanto que $g(0,0,\ldots,0) = 0$, e portanto

$$\Pr_y[C'(y) \neq g(y)] \geq \Pr_y[y = (0,0,\ldots,0)] = (1-2^{-b})^{2^b} = 1/e > 1/4 > 0,2,$$

contradizendo a nossa hipótese. Portanto, deve haver uma cláusula que consiste somente de literais positivos e a probabilidade que esta cláusula é verdadeira é no máximo $w'2^{-b}$, com isto dito:

$$\Pr_y[C'(y) = 0] \geq 1 - w'2^{-b}.$$

E como C' 0,2-aproxima a função g devemos ter $1 - w'2^{-b} \leq 1/e + 0,2 \Rightarrow w' \geq (0,8 - 1/e)2^b \geq (1/3)2^b$.

□

Note que por dualidade, o mesmo limite inferior vale para a função $\text{Tribes}^\dagger_{b.2^b}$.

Corolário 3.19. *Seja C um circuito FNC que 0,2-aproxima a função $\text{Tribes}^\dagger_{b,2^b}$ com largura w. Então devemos ter $w \geq (1/3)2^b = \Omega(m/\log m)$.*

Agora podemos provar o teorema 3.17.

Demonstração. (Prova do teorema 3.17)

Primeiro nos consideraremos o caso que o circuito C está na FND. Por (...) sabemos que existe um circuito FND de largura $\log(100s)$ que $0,04 + 1/100 = 0,05$-aproxima F_m. Seja $x \sim \text{Tribes}^{-1}_{b,2^b}(0)$ e $y \sim \{0,1\}^m$. Então temos que

$$\begin{aligned}
\Pr_{x\sim\text{Tribes}^{-1}_{b,2^b}(0),y}[C(x,y) \neq F_m(x,y)] &= \Pr_{x,y}[C(x,y) \neq F_m(x,y) \big| x \in \text{Tribes}^{-1}_{b,2^b}(0)] \\
&= \frac{\Pr_{x,y}[C(x,y) \neq F_m(x,y) \wedge x \in \text{Tribes}^{-1}_{b,2^b}(0)]}{\Pr_x[x \in \text{Tribes}^{-1}_{b,2^b}(0)]} \\
&\leq 4 \Pr_{x,y}[C(x,y) \neq F_m(x,y)] \\
&\leq 4 \times 0,05 = 0,2.
\end{aligned}$$

Donde nós usamos o fato que $\Pr[a \wedge b] \leq \Pr[a]$, para quaisquer eventos a e b, e também que pelo menos uma fração de $1/4$ das strings $x \in \{0,1\}^m$ fazem a função $\text{Tribes}_{b,2^b}(x) = 0$ na primeira desigualdade.

Mas quando $\text{Tribes}_{b,2^b}(x) = 0$, $F_m(x,y)$ é simplesmente $\text{Tribes}^{\dagger}_{b,2^b}(y)$. Então podemos fixar um x^* tal que

$$\Pr_{y\sim\{0,1\}^m}[C(x^*,y) \neq \text{Tribes}^{\dagger}_{b,2^b}(y)] < 0,2.$$

Seja C' o circuito tal que $C'(y) = C(x^*,y) = \text{Tribes}^{\dagger}_{b,2^b}(y)$. Como a largura de C' é no máximo $\log(100s)$, pelo corolário 3.19 devemos ter

$$\begin{aligned}
\log(100s) &\geq (1/3)2^b \\
100s &\geq 2^{(1/3)2^b} \\
s &\geq 2^{(1/3)2^b}/100.
\end{aligned}$$

E como $2^{(1/3)2^b}/100 = 2^{\Omega(m/\log m)}$, nós completamos a prova para o caso em que C é um circuito FND. No caso em que C é um circuito FNC nós meio que trocamos o papel das funções $\text{Tribes}_{b,2^b}$ e $\text{Tribes}^{\dagger}_{b,2^b}$ e usamos os mesmos argumentos.

□

Só como um comentário, o argumento que O'Donnell e Wimmer usaram para provar o teorema 3.18 é basicamente um dos últimos passos que (...) usaram para provar (...). Lembrando que eles queriam mostrar que a função Sipser'_d não pode ser aproximada por circuitos de profundidade $d-1$ e tamanho subexponencial. A estratégia deles foi definir um operador Ψ de forma que com alta probabilidade $\Psi(\mathsf{Sipser}')$ pode ser representada como Or_S, o "ou" sobre variáveis $\{y_i\}_{i\in S}$, enquanto que $\Psi(C)$ é representada por um uma fórmula FNC com largura "pequena" sobre as mesmas variáveis $\{y_i\}_{i\in S}$ e portanto não pode ser o caso que C esteja perto de Sipser'.

3.5 A hipótese dos oráculo aleatório

Capítulo 4

Restrições e projeções aleatórias

Nós já vimos no capítulo anterior que a busca por separações de algumas classes de complexidade no mundo relativizado motivou a pesquisa em complexidade de circuitos. Em especial, nós vimos como alguns limitantes inferiores para classes de circuito que somente admitem circuitos de profundidade constante implicariam no colapso de algumas classes de complexidade relativas a algum oráculo. Agora iremos ver como provar estes limitante inferiores, concluindo então as provas dos teoremas do capítulo 3.

Na seção 4.1 iremos provar o resultado de (...) que as funções paridades que definimos em 3.7 não têm circuitos de profundidade constante e tamanho subexponencial. Este resultado pode também ser expresso como Parity $\notin \mathsf{AC}^0$.

Também, como foi visto em 3.4, nos interessa provar que existe uma hierarquia de circuitos de profundidade constante. Ou seja, que circuitos de profundidade d são estritamente mais fortes do que circuitos de profundidade $d-1$ para todo $d \geq 2$. Se denotarmos por AC^0_d a classe AC^0 restrita a circuitos de profundidade d então a existência de uma hierarquia de profundidade implicaria em $AC^0_d \nsubseteq AC^0_{d-1}$, para todo $d \geq 2$. Este resultado foi provado pela primeira vez por Håstad em [Hås87]. Na seção 4.2 nós iremos ver a prova de Rossman, Servedio e Tan [RST15a] que também nos dá uma hierarquia de profundidade constante até mesmo quando apenas exigimos que circuitos de profundidade $d-1$ sejam capazes de aproximar circuitos de profundidade d.

Um componente chave das estratégias adotada nas duas seções deste capítulo envolve aleatoriamente restringir algumas variáveis de entrada do circuito com o objetivo de obter um circuito mais simples do que o original. Esta técnica é atribuida à Subbotovskaya que a introduziu em [Sub61]. Nós iremos ver que para provar que circuitos simplificam ao aplicarmos uma restrição aleatória sobre suas variáveis de entrada nós precisamos de um lema da troca que prova o "caso base" que fórmulas FNCs e FNDs com alta probabilidade simplificam ao serem atingidas por restrições aleatórias. Em [Bea94], Beame discute algumas aplicações de lemas da troca.

4.1 Restrições aleatórias e a prova de Håstad dos teoremas 3.8 e 3.9

De maneira geral, uma restrição a um conjunto de variáveis $X = \{x_i\}_{i\in[n]}$ é um mapeamento $\rho : X \to \{*, 0, 1\}^n$. Se $\rho(x_i) = *$ dizemos que x_i é uma variável livre. Se f é uma função sobre as variáveis X e aplicamos uma restrição ρ sobre X, obtemos uma nova função $f_{|\rho}$ sobre as variáveis em $\rho^{-1}(*)$. Se $f : \{0,1\}^n \to \{0,1\}$, então temos que

$$f_{|\rho}(x_1, x_2, \ldots, x_n) = f(val_\rho(x_1), val_\rho(x_2), \ldots, val_\rho(x_n)),$$

em que $val_\rho(x_i)$ é 0 ou 1 se $\rho(x_i)$ é 0 ou 1, respectivamente, e $val_\rho(x_i) = x_i$ caso contrário.

Como já discutimos, ao tentar provar limitantes inferiores restrições são interessantes pois elas simplificam circuitos. Em particular, se considerarmos um circuito C de profundidade constante temos que $C_{|\rho}$ pode acabar sendo uma constante, ou uma função representável por uma árvore de decisão de profundidade pequena. Para que possarmos ser mais concretos, nós iremos agora dar uma definição formal para restrições aleatórias.

Definição 4.1. *(Restrições aleatórias)*

Seja $p \in (0, 1]$, uma restrição aleátoria $\boldsymbol{\rho}$ sobre as variáveis $\{x_i\}_{i\in[n]}$ é tirada da seguinte distribuição R_p. Para cada $i \in [n]$,

$$\boldsymbol{\rho}(x_i) = \begin{cases} * & \textit{com probabilidade } p \\ 0 & \textit{com probabilidade } (1-p)/2 \\ 1 & \textit{com probabilidade } (1-p)/2. \end{cases}$$

Quando $\boldsymbol{\rho}$ for tirada de R_p escreveremos $\boldsymbol{\rho} \leftarrow R_p$. Se $\rho^{-1}(*) = \emptyset$ nós dizemos que ρ é uma atribuição.

A idéia é que quando aplicamos uma restrição $\boldsymbol{\rho} \leftarrow R_p$, para algum valor $p \in (0, 1]$ bem próximo de 0, sobre as variáveis de entrada de um circuito C, com alta probabilidade este circuito irá degenerá-se em uma função extremamente simples (por exemplo, uma árvore de decisão de baixa profundidade). Daí, se houver uma função f que provadamente não simplifica sobre uma restrição $\boldsymbol{\rho} \leftarrow R_p$ (i.e., mantém algum tipo de estrutura) poderemos concluir que o circuito C não pode computar a função f.

Como exemplo de funções que não simplifica ao ter uma restrição tirada de R_p aplicada às suas variáveis de entrada, nós podemos tomar as funções Parity_n. Nós veremos que ao aplicarmos uma restrição $\boldsymbol{\rho} \leftarrow R_p$ sobre as n variáveis de entrada de Parity_n obteremos uma função sobre n' variáveis, em que $E[n'] = pn$, que é somente a função $\text{Parity}_{n'}$ ou a função $1 - \text{Parity}_{n'}$. Em particular, n' irá crescer em expectativa enquanto n também cresce. Assim, podemos provar que com probabilidade muito alta as funções Parity_n não são representáveis por árvores de decisão de profundidade constante quando n é suficientemente grande, pois também é verdade que as funções paridades são evasivas[1].

O lema da troca de Håstad

Nós queremos provar o teorema 3.8. Para isso, nós provaremos que se fizermos p pequeno o suficiente, então para algum circuito C de profundidade constante uma restrição $\boldsymbol{\rho} \leftarrow R_p$ fará o circuito $C_{|\boldsymbol{\rho}}$ computar uma função representável por uma árvore de decisão de profundidade constante. Note que isso é o suficiente para provar que C não poderia computar Parity_n porque quando restringimos as variáveis de entrada de Parity_n de forma que um número não tão pequeno de variáveis permaneçam livre, nós simplesmente obtemos uma nova função paridade que por sua vez não pode ser computada por árvores de decisão com profundidade pequena.

Então, para provar o teorema 3.8 precisamos antes provar que circuitos simplificam após uma restrição, e para isso usamos o lema de Håstad.

Lema 4.2. *(Lema de Håstad)*

Seja F uma fórmula FNC (ou FND) com largura w, $s \geq 1$, $p \in (0, 1]$ e $\boldsymbol{\rho} \leftarrow R_p$, então

$$\Pr[D(F_{|\boldsymbol{\rho}}) \geq s] \leq (5pw)^s$$

[1] Como já foi dito na seção 3.1 uma função sobre n variáveis é dita ser evasiva se ela não pode ser representável por uma árvore de decisão com profundidade menor do que n

Nós vamos ver 2 provas diferente do lema 4.2. A primeira é a prova original do próprio Håstad [Hås87]. A segunda prova é de Razborov e aparece em [Bea94].

Na primeira prova nós iremos considerar $F = \bigwedge_{i=1}^{m} C_i$ e iremos argumentar por indução em m, o número de cláusulas na prova. Na verdade, nós provamos o lema 4.6 que é uma versão mais forte do lema de Håstad. Antes precisamos das seguinte definições.

Definição 4.3 (Extensão de uma restrição)**.** *Seja $\rho \in \{*, 0, 1\}^n$ uma restrição às variáveis $\{x_i\}_{i\in[n]}$. Uma restrição ρ' é dita ser uma extensão de ρ se e somente $\rho'^{-1}(*) \subseteq \rho^{-1}(*)$.*

Ou seja, ρ' é uma extensão de ρ se todas as variáveis que foram fixadas como uma constante em ρ também são fixadas como uma constante em ρ', e adicionalmente ρ' pode ter fixado algumas variáveis a mais como constantes.

Sendo assim podemos até mesmo definir uma ordem parcial sobre as restrições em $\{*, 0, 1\}^n$ em que $\rho \leq \rho'$ se e somente ρ' é uma extensão de ρ. Neste caso temos que atribuições são os elementos maximais deste ordenamento.

Definição 4.4 (Conjuntos inferiores)**.** *Seja Δ um conjunto de restrições em $\{*, 0, 1\}^n$. Nós dizemos que Δ é um conjunto inferior se para toda restrição $\rho \in \Delta$ e toda extensão ρ' de ρ é verdade que $\rho' \in \Delta$.*

Em outras palavras, se Δ for um conjunto inferior e $\rho \in \Delta$ então restringir ainda mais variáves de ρ resultará numa restrição que ainda está em Δ.

Uma cadeia é um conjunto de restrições tal que o ordenamento induzido pelos elementos deste conjunto é um ordenamento total. Nós precisamos ainda da seguinte definição.

Definição 4.5 (Subconjunto inferior maximal)**.** *Seja S um conjunto qualquer de restrições em $\{*, 0, 1\}^n$. O subconjunto inferior maximal de S, que iremos denotar por S^0 é a união de todos subconjuntos de S que são cadeias e contêm um elemento maximal – equivalentemente, contêm uma atribuição.*

Nós podemos notar que

- S^0 é o maior conjunto inferior contido em S.
- S^0 pode ser obtido removendo todas restrições ρ em S tais que existe uma extensão ρ' de ρ e $\rho' \notin S$.

Agora podemos enunciar a versão mais forte do lema de Håstad que iremos provar.

Lema 4.6. *Seja F uma fórmula FNC (ou FND) com largura w, $s \geq 1$, $p \in (0, 1]$ e $\boldsymbol{\rho} \leftarrow R_p$, então*

$$\Pr[D(F|_{\boldsymbol{\rho}}) \geq s | \boldsymbol{\rho} \in \Delta] \leq (5pw)^s,$$

onde Δ é um conjunto inferior arbitrário.

O lema 4.6 é uma versão mais forte do lema de Håstad pois o conjunto de todas restrições em $\{*, 0, 1\}^n$ é um conjunto inferior. Porém, iremos precisar da condicionate para que o argumento por indução funcione.

Demonstração. (Primeira prova do lema de Håstad)

Como já foi dito iremos considerar $F = \bigwedge_{i=1}^{m} C_i$, onde cada cláusula C_i tem largura no máximo w. Nós iremos argumentar por indução em m, o número de cláusulas em F. Nós iremos assumir ao longo da prova que $5pw < 1$, pois caso contrário o lema é trivial.

Se $m = 0$ então $F = 1$ e não precisamos mais fazer nada. Suponha agora que $m > 0$, nós então temos que

$$\Pr[D(F_{|\rho}) \geq s|\rho \in \Delta] \leq \max(\Pr[D(F_{|\rho}) \geq s|\rho \in \Delta \wedge C_{1|\rho} = 1], \Pr[D(F_{|\rho}) \geq s|\rho \in \Delta \wedge C_{1|\rho} \neq 1]),$$

em que Δ é um conjunto inferior qualquer. Desta forma apenas nos basta mostrar que o lema é verdadeiro em ambos os casos ($C_{1|\rho} = 1$ e $C_{1|\rho} \neq 1$). Vamos primeiro considerar o caso em que $C_{1|\rho} = 1$. Se $C_{1|\rho} = 1$ então $F_{|\rho} = \bigwedge_{i=2}^{m} C_{i|\rho}$, e iremos chamar F sem a sua primeira cláusula de F' (daí temos que $F_{|\rho} = F'_{|\rho}$ quando $C_{1|\rho} = 1$). Temos que

$$\Pr[D(F_{|\rho}) \geq s|\rho \in \Delta \wedge C_{1|\rho} = 1] = \Pr[D(F'_{|\rho}) \geq s|\rho \in \Delta \wedge C_{1|\rho} = 1].$$

Como F' é uma fórmula FNC com $m - 1$ cláusulas podemos usar a hipótese da indução dado que a condicionante nas probabilidades acima induz um conjunto inferior. Mas isto segue do seguinte fato:

Fato 4.7. *Se Δ_1 e Δ_2 são conjuntos inferiores então $\Delta_1 \cap \Delta_2$ é um conjunto inferior de restrições.*

E como o conjunto de todas as restrições que satisfazem a primeira cláusula é um conjunto inferior, segue que

$$\Pr[D(F_{|\rho}) \geq s|\rho \in \Delta \wedge C_{1|\rho} = 1] \leq (5pw)^s.$$

Agora iremos o considerar o caso em que $C_{1|\rho} \neq 1$. Primeiro observamos que se $C_{1|\rho} = 0$ então $F_{|\rho} = 0$ e o resultado é trivial pois a função 0 tem uma árvore de decisão de profundidade 0. Iremos assumir então que $C_{1|\rho} \neq 0$. Agora não temos mais que $C_{1|\rho} \neq 1$ representa um conjunto inferior, portanto teremos que fazer algum esforço antes de poder usar a hipótese da indução.

Seja T o conjunto das variáveis que aparecem em C_1. Também, seja ρ uma restrição tal que $C_{1|\rho} \neq 1$ e $D(F_{|\rho}) \geq s$. Como estamos assumindo que $C_{1|\rho} \neq 0$ temos que deve haver um subconjunto Y de T tal que $\rho(x_i) = *$, para todo $x_i \in T$, e $\rho(x_j) \in \{0, 1\}$ para todo $x_j \in T \setminus Y$. Iremos chamar este evento de $\rho(Y) = *$. Para cada atribuição π às variáveis em Y nós definimos o conjunto X_π da seguinte forma.

$$X_\pi = \{\rho' | \rho' = \rho\pi \text{ para alguma restrição } \rho \text{ e } C_{1|\rho} \neq 1\}.$$

O conjunto que realmente nos interessa é o subconjunto inferior maximal de X_π que denotamos por X^0_π. Nós podemos notar que $\rho' \in X^0_\pi$ se e somente se $\rho' = \rho\pi$ e ρ é tal que $\rho(Y) = *$. Temos a seguinte identidade.

$$\Pr[D(F_{|\boldsymbol{\rho}}) \geq s|\boldsymbol{\rho} \in \Delta \wedge C_{1|\boldsymbol{\rho}} \neq 1] = \sum_{Y \subseteq T, Y \neq \emptyset} \Pr[\boldsymbol{\rho}(Y) = *|\boldsymbol{\rho} \in \Delta \wedge C_{1|\boldsymbol{\rho}}] \times \Pr[D(F|\boldsymbol{\rho}) \geq s|\boldsymbol{\rho} \in \Delta \wedge C_{1|\boldsymbol{\rho}} \neq 1 \wedge \boldsymbol{\rho}(Y) = *]. \tag{4.1}$$

Então vamos limitar o valor de ambos os fatores que aparecem em cada termo no somatório do lado direito de 4.1. Primeiro argumentamos que

$$\Pr[\rho(Y) = *|\rho \in \Delta \wedge C_{1|\rho} \neq 1] \leq \left(\frac{2p}{1+p}\right)^{|Y|}. \tag{4.2}$$

Dizer que $C_{1|\rho} \neq 1$ é o mesmo que dizer que $\rho(x_i) \in \{b, *\}$, para cada variável $x_i \in T$ em que b é 0 se x_i aparece não-negada em C_1 e $b = 1$ caso contrário. Então dado que $\rho(x_i) \neq b$ temos que $\rho(i)$ é * com probabilidade $\frac{p}{1-\frac{1-p}{2}} = \frac{2p}{1+p}$, e como ρ atribui valores às variáveis de forma independente obtemos 4.2.

Para estimar o segundo fator nós precisamos do seguinte fato.

Fato 4.8. *Se ρ é tal que $D(F_{|\rho}) \geq s$ e $\rho(Y) = *$, então existe uma atribuição π às variáveis em Y tal que $C_{1|\pi} = 1$ e $D(F_{|\rho\pi}) \geq s - |Y|$.*

Isso é verdade pois se para todas tais atribuições π fosse verdade que $D(F_{|\rho\pi}) < s - |Y|$, então poderiamos construir uma árvore de decisão para $F_{|\rho}$ que primeiro faz consultas à todas as variáveis de Y e depois usa a árvore de decisão de profundidade menor do que $s - |Y|$ para $F_{|\rho\pi}$, em que π é a atribuição consistente com as respostas das consultas da árvore de decisão. Esta árvore de decisão teria profundidade menor do que $(s - |Y|) + |Y| = s$, o que é uma contradição. Além disso, a única atribuição π_{falsa} às variáveis de Y que torna a cláusula C_1 falsa ao ser composta com ρ satisfaz $F_{|\rho\pi_{\text{falsa}}} = 0$ e portanto ela não pode ser a atribuição que contradiz a asserção $D(F_{|\rho\pi}) < s - |Y|$ para todas atribuições π.

Então, pelo princípio da inclusão-exclusão, nós temos a seguinte desigualdade.

$$\Pr[D(F_{|\boldsymbol{\rho}}) \geq s | \boldsymbol{\rho} \in \Delta \wedge C_{1|\boldsymbol{\rho}} \neq 1 \wedge \boldsymbol{\rho}(Y) = *] \leq \sum_{\pi} \Pr[D(F_{|\boldsymbol{\rho}\pi}) \geq s - |Y| | \boldsymbol{\rho} \in \Delta \wedge C_{1|\boldsymbol{\rho}} \neq 1 \wedge \boldsymbol{\rho}(Y) = *], \quad (4.3)$$

em que a soma no lado direito é somente sobre as atribuições π que tornam C_1 verdadeira. Mas agora temos que $C_{1|\rho} \neq 1$ e $\rho(Y) = *$ se e somente se $\rho\pi \in X^0_\pi$. Nos podemos então expressar a desigualdade 4.3 da seguinte forma.

$$\Pr[D(F_{|\boldsymbol{\rho}}) \geq s | \boldsymbol{\rho} \in \Delta \wedge C_{1|\boldsymbol{\rho}} \neq 1 \wedge \boldsymbol{\rho}(Y) = *] \leq \sum_{\pi} \Pr[D(F_{|\boldsymbol{\rho}\pi}) \geq s - |Y| | \boldsymbol{\rho} \in \Delta \wedge \boldsymbol{\rho}\pi \in X^0_\pi]. \quad (4.4)$$

Pelo subconjunto inferior maximal de um conjunto ser certamente um conjunto inferior e o fato 4.7, temos que agora há um conjunto inferior expresso na condicionante [2]. Além disso, como $F_{|\rho\pi}$ não depende da primeira cláusula de F, nós por fim podemos usar a hipótese indutiva para deduzir que para todo π que estamos levando em consideração é verdade que

$$\Pr[D(F_{|\boldsymbol{\rho}\pi}) \geq s - |Y| | \boldsymbol{\rho} \in \Delta \wedge \boldsymbol{\rho}\pi \in X^0_\pi] \leq (5pw)^{s-|Y|}. \quad (4.5)$$

Lembrando que por estarmos assumindo que $5pw < 1$ nós podemos até mesmo assumir que $|Y| < s$ sem perda de generalidade. Usando que há $2^{|Y|} - 1$ atribuições às variáveis de Y que tornam C_1 verdadeira, nós podemos reescrever a desigualdade 4.4 como

$$\Pr[D(F_{|\rho}) \geq s | \rho \in \Delta \wedge C_{1|\rho} \neq 1 \wedge \rho(Y) = *] \leq (2^{|Y|} - 1)(5pw)^{s-|Y|}. \quad (4.6)$$

Juntando 4.1, 4.2 e 4.6 temos que

$$\Pr[D(F_{|\rho}) \geq s | \rho \in \Delta \wedge C_{1|\rho} \neq 1] \leq \sum_{Y \subseteq T, Y \neq \emptyset} \left(\frac{2p}{1+p}\right)^{|Y|} (2^{|Y|} - 1)(5pw)^{s-|Y|}.$$

Rearranjando os termos do somatório pela cardinalidade do conjunto Y nós temos que

[2] Bem, na verdade temos que condicionar em $\boldsymbol{\rho}$ pertencer a um conjunto inferior. Mas o conjunto de restrições ρ tais que $\rho\pi \in X^0_\pi$ ainda é um conjunto inferior.

$$\begin{aligned}\Pr[D(F_{|\boldsymbol{\rho}}) \geq s | \boldsymbol{\rho} \in \Delta \wedge C_{1|\boldsymbol{\rho}} \neq 1] &\leq \sum_{k=0}^{|T|} \binom{|T|}{k} \Big(\frac{2p}{1+p}\Big)^k (2^k - 1)(5pw)^{s-k} \\ &= (5pw)^s \sum_{k=0}^{|T|} \binom{|T|}{k} \Big(\frac{2}{5w(1+p)}\Big)^k (2^k - 1) \\ &= (5pw)^s \bigg(\Big(1 + \frac{4}{5w(1+p)}\Big)^{|T|} - \Big(1 + \frac{2}{5w(1+p)}\Big)\bigg).\end{aligned}$$

Usando que $(1+2x) \leq (1+x)^2$ e $(1+x) \leq e^x$ nós obtemos que

$$\Pr[D(F_{|\boldsymbol{\rho}}) \geq s | \boldsymbol{\rho} \in \Delta \wedge C_{1|\boldsymbol{\rho}} \neq 1] \leq (5pw)^s \big(e^{4p|T|/5w(1+p)} - e^{2p|T|/5w(1+p)}\big).$$

Usando que $|T| \leq w$ e notando que $e^{4/2} - e^{2/5} < 1$ nós podemos finalmente concluir que

$$\Pr[D(F_{|\rho}) \geq s | \rho \in \Delta \wedge C_{1|\rho} \neq 1] \leq (5pw)^s.$$

Então provamos que em ambos os casos em que ρ faz a primeira cláusula de F ser 1 ou não ser 1 a probabilidade que $D(F_{|\rho}) \geq s$ é no máximo $(5pw)^s$, concluindo então esta prova do lema de Håstad.

□

Prova do Lema da troca de Håstad usando o método de Razborov

Agora iremos ver a segunda prova do Lema de Håstad. Primeiro nós vamos definir o que é a árvore de decisão canônica de uma fórmula FNC F.

Definição 4.9. *(Árvore de decisão canônica de F)*

Seja F uma fórmula FNC, a árvore de decisão canônica T de F é definida recursivamente da seguinte forma:

1. *Se $F = 0$ ou $F = 1$ então T é simplesmente a árvore de decisão trivial que não faz nenhuma consulta e tem uma única folha com o valor apropriado.*
2. *Seja C_1 a primeira cláusula não vazia de F e K o conjunto das variáveis que aparecem em C_1. Então T faz consultas às variáveis em K em alguma ordem arbitrária, formando em cada caminho uma folha associada à uma restrição ρ que seta os valores de K de forma consistente com o caminho da raiz de T até esta folha. Por fim trocamos cada folha com a árvore de decisão canônica de $F_{|\rho}$, em que ρ é a restrição associada à folha.*

Definição 4.10. *A complexidade de consulta canônica de F que denotaremos por $D_{can}(F)$ é a profundidade da árvore de decisão canônica de F.*

A idéia agora é codificar restrições "más", significando restrições $\rho \in \{*, 0, 1\}^n$ tais que $D_{\text{can}}(F_{|\rho}) \geq s$, de forma que o mapeamento de restrições más para códigos seja injetivo. Depois disso mostramos que o número de possíveis códigos é pequeno dando então um limitante superior para o número de restrições más.

Lema 4.11. *Seja F uma fórmula FNC (ou FND) com largura w, $s \geq 1$, $0 < p \leq 1/5$ e $\rho \leftarrow R_p$, então*

$$\Pr[D_{can}(F_{|\boldsymbol{\rho}}) \geq s] \leq (5pw)^s.$$

Note que adicionamos a restrição $p \leq 1/5$, o que todavia não tem nenhum impacto em qualquer aplicação do Lema da troca de Håstad que aparece neste trabalho. Porém, como veremos, não podemos remover tal restrição sem aumentar a constante que aparece no lado direito da desigualdade no enunciado do lema.

Demonstração. (Segunda prova do lema de Håstad)

Vamos considerar $F = \bigwedge_{i=1}^{m} C_i$ onde cada cláusula C_i tem largura no máximo w. Como já foi dito, o objetivo é codificar todas as restrições $\rho \in \{*, 0, 1\}^n$ que sejam más. Seja $\mathcal{B}$ o conjunto de todas as restrições $\rho \in \{*, 0, 1\}^n$ tais que $D_{\text{can}}(F_{|\rho}) \geq s$. O nosso mapeamento é

$$\begin{aligned} \text{Code} : \mathcal{B} &\to \{*, 0, 1\}^n \times [w]^s \times \{0, 1\}^s \\ \text{Code}(\rho) &\mapsto (\rho', \beta, \delta). \end{aligned}$$

Em que ρ' é uma extensão de ρ e β e δ são informações adicionais que serão importante no processo de recuperar ρ a partir do seu código. Note que como $p \leq 1/5 < 1/2$ temos que qualquer extensão de uma restrição ρ terá um peso maior do que ρ no espaço de restrições R_p. Nós queremos que o mapeamento seja injetivo para que possamos usar o fato que ρ' tem um peso maior do que ρ para provar um limitante superior para a probabilidade $\Pr[\boldsymbol{\rho} \in \mathcal{B}]$.

Nós iremos dividir o restante da prova em três partes. Na primeira parte iremos descrever o processo de construir a partir de $\rho \in \mathcal{B}$ um código da forma (ρ', β, δ). Na segunda parte nós descrevemos em detalhe como recuperar ρ a partir de seu código. E finalmente na terceira parte nós estimamos a probabilidade que uma restrição $\rho \leftarrow R_p$ está no conjunto $\mathcal{B}$.

1. Construindo o código a partir de ρ.

   Seja $\rho \in \mathcal{B}$. Seja π um caminho qualquer da raíz até uma folha da árvore de decisão canônica T de $F_{|\rho}$ que tenha tamanho $\geq s$, truncando π de tal forma que $|\pi| = s$. Nós iremos ver π como uma atribuição natural às variávels que foram consultadas por T no caminho π. Nós construimos o código em estágio. Pra começar descrevemos o primeiro passo. Seja C_{α_1} a primeira cláusula da fórmula $F_{|\rho}$ que não seja igual a 1 (tal cláusula deve existir porque $F_{|\rho} \neq 1$) e chame de K_1 o conjunto de variáveis que aparecem em $C_{\alpha_1|\rho}$. Seja π_1 a parte de π que contém variáveis em K_1. Seja σ_1 a única atribuição às variáveis em K_1 que não satisfaz a cláusula $C_{\alpha_1|\rho}$. Consideramos então o vetor $\beta_1 \in [w]^{|K_i|}$ tal que a j-ésima coordenada de β_1 indica a posição da j-ésima variável em K_i na clásula C_i (nós podemos assumir alguma enumeração arbitrária das variáveis em cada cláusula de F).

   Para $i > 1$ nós fazemos C_{α_i} ser a primeira cláusula na fórmula $F_{|\rho\pi_1\pi_2\ldots\pi_{i-1}}$ que não é igual a 1 e definimos σ_i, β_i e π_i de forma análoga a como fizemos no primeiro passo. Nós chegamos no último estágio quando $\pi_i = \pi \setminus \pi_1 \ldots \pi_{i-1}$. Vamos assumir que k estágios foram realizados, para algum $k \geq 1$ e notamos que a restrição π_k pode não setar valores à todas variáveis na cláusula $C_{\alpha_k|\rho\pi_1\ldots\pi_{k-1}}$ – em outras palavras, é possível que σ_k não tenha atribuido valores à todas as variáveis ainda vivas em $C_{\alpha_k|\rho\pi_1\ldots\pi_{k-1}}$ – pois truncamos o caminho π da árvore de decisão canônica de forma que ela contenha exatamente s variáveis. Porém, como veremos, necessitamos apenas que esta cláusula não seja satisfeita pela atribuição σ_k.

   Seja $\sigma = \sigma_1\sigma_2\ldots\sigma_k$ e $\beta = \bigcup_{i=1}^{k} \beta_i$. Por fim definimos a string $\delta \in \{0, 1\}^s$ em que a i-ésima coordenada de d é 1 se e somente se a i-ésima variável setada pela restrição σ (seguindo algum ordenamento das cláusulas de F e das variáveis dentro dessas cláusulas) tem um valor atribuido diferente nas restrições σ e π.

   A extensão de ρ que aparece no seu código é $\rho' = \rho\sigma = \rho\sigma_1\sigma_2\ldots\sigma_k$. Nós então fazemos $\text{Code}(\rho) = (\rho', \beta, \delta)$.

2. Decodificando Code(ρ).

   Dado Code(ρ) $= (\rho\sigma_1\sigma_2\ldots\sigma_k, \beta, \delta)$ nós recuperamos ρ em estágios. Nós assumimos que no i-ésimo estágio nós já recuperamos as restrições $\pi_1, \pi_2, \ldots, \pi_{i-1}$. Lembrando o i-ésimo estágio do processo de construção de Code(ρ) nós tinhamos que C_{α_i} era a primeira cláusula que não era igual a 1 na fórmula $F_{|\rho\pi_1\pi_2\ldots\pi_{i-1}}$, e o mesmo vale para a fórmula $F_{|\rho\pi_1\pi_2\ldots\pi_{i-1}\sigma_i\ldots\sigma_k}$ pois cada σ_j, para $j > i$, só atribui valores a variáveis que não aparecem em $C_{\alpha_i|\rho\pi_1\pi_2\ldots\pi_{i-1}}$ e também σ_i foi definida de forma que $C_{\alpha_i|\rho\pi_1\pi_2\ldots\pi_{i-1}\sigma_i}$ não é igual a 1. Isso significa que a partir de $\rho\pi_1\pi_2\ldots\pi_{i-1}\sigma_i\ldots\sigma_k$ podemos encontrar C_{α_i}. Agora podemos recuperar π_i a partir de C_{α_i}, β_i e a string δ. Agora, dado que sabemos cada π_i podemos dizer qual parte da restrição $\rho\pi_1\pi_2\ldots\pi_k$ é ρ: ρ só atribui valores à variáveis que não aparece em nenhum dos π_i (ou σ_i).

3. Estimando $\Pr[\boldsymbol{\rho} \in \mathcal{B}]$.

   Seja $\rho \in \mathcal{B}$ e a o número de variáveis feitas constante por ρ. Então temos o seguinte:

$$\frac{\Pr[\boldsymbol{\rho} = \rho]}{\Pr[\boldsymbol{\rho} = \rho']} = \frac{p^{n-a}(1-p)^a}{p^{n-a-s}(1-p)^{a+s}} = \left(\frac{p}{1-p}\right)^s,$$

   em que n é o número total de variáveis na fórmula e ρ' é a extensão de ρ que aparece em seu código. Portanto,

$$\Pr[\boldsymbol{\rho} = \rho] = \left(\frac{p}{1-p}\right)^s \Pr[\boldsymbol{\rho} = \rho'].$$

   Como estamos assumindo que $p \leq 1/5$:

$$\Pr[\boldsymbol{\rho} = \rho] \leq \left(\frac{5p}{4}\right)^s \Pr[\boldsymbol{\rho} = \rho'].$$

   Seja $\mathcal{C}$ o conjunto de todas as possíveis extensões ρ' das restrições $\rho \in \mathcal{B}$. Usando o fato que Code é um mapeamento injetivo e que β pode ser representado por uma string em $\{0,1\}^{s(\log w+1)}$ enquanto que d é representável por uma string em $\{0,1\}^s$:

$$\begin{aligned}
\Pr[\boldsymbol{\rho} \in \mathcal{B}] &= \sum_{\rho\in\mathcal{B}} \Pr[\boldsymbol{\rho} = \rho] \\
&\leq 2^{s(\log w+1)} 2^s \left(\frac{5p}{4}\right)^s \sum_{\rho'\in\mathcal{C}} \Pr[\boldsymbol{\rho} = \rho'] \\
&\leq (5pw)^s,
\end{aligned}$$

   onde na última desigualdade nós usamos que a soma de probabilidades é no máximo 1.

□

Prova dos teoremas 3.8 e 3.9

Agora nós podemos ver como uma restrição aleatória simplifica um circuito C de profundidade d. Suponha que as portas no primeiro nível de C sejam portas $\wedge$, e portanto temos portas $\vee$ no segundo nível e portas $\wedge$ no terceiro nível. As portas $\vee$ no segundo nível computam uma fórmula FND e portanto se aplicarmos uma restrição às variáveis de entrada desta fórmula FND, com alta probabilidade podemos trocar ela por uma

árvore de decisão, o que também pode ser vista como uma fórmula FNC. Se fizermos o mesmo com todas as portas $\vee$ no segundo nível, teremos somente portas $\wedge$ no segundo nível alimentando portas $\wedge$ no terceiro nível. Então podemos absorver as portas $\wedge$ no segundo nível diminuindo a profundidade do circuito em 1.

Nós iremos usar este argumento para provar 3.8.

Teorema 4.12. *Seja $d > 0$ um inteiro. Para n suficientemente grande temos que qualquer circuito de profundidade d com fan-in* $\mathsf{polylog}(n)$ *no seu primeiro nível e tamanho $< 2^{\mathcal{O}\left(n^{\frac{1}{d-1}}\right)}$ não pode computar a função paridade de n variáveis corretamente em todas as entradas.*

Demonstração. (Prova do teorema 3.8)

Nós mostraremos que C e Parity_n com alta probabilidade colapsam para funções diferentes quando nós aplicamos uma restrição $\rho \leftarrow R_p$ onde

$$p = \frac{1}{10w}\left(\frac{1}{10\log(120S)}\right)^{d-2}.$$

Especificamente, nós iremos mostrar que:

1. Com probabilidade maior do que 99%, $C_{|\rho}$ tem uma árvore de decisão de profundidade no máximo 10.
2. Com probabilidade maior do que 99%, $\text{Parity}_{n|\rho}$ é a função paridade ou a negação da função paridade de mais do que 10 variáveis.

Provar (1) e (2) é suficiente para provar o teorema porque a função paridade e a sua negação são funções evasivas, e portanto depender de mais do que 10 variáveis implica em não ter uma árvore de decisão de profundidade no máximo 10. Então temos que com probabilidade maior do que $1 - 2 \times 0,01 = 0,98$, $C_{|\rho} \neq \text{Parity}_{n|\rho}$ o que implica em C não poder ser um circuito para a função Parity_n.

Provando (1):

Nós usamos o fato que aplicar uma restrição $\rho \leftarrow R_p$ à uma função é o mesmo que aplicar uma sequência de restrições $\rho_1, \rho_2, \ldots \rho_{d-1}$, onde $\rho_i \leftarrow R_{p_i}$ onde p_i é $\frac{1}{10w}$ quando i = 1 e $p_i = \frac{1}{10\log(120S)}$ para $2 \leq i \leq d-1$. Em cada aplicação das restrições ρ_i nós usamos o Lema de Håstad para mostrar que com alta probabilidade o circuito $C_{|\rho_1\rho_2\ldots\rho_{i-1}}$ tem sua profundidade diminuida, por fim teremos que, com alta probabilidade, $C_{|\rho_1\rho_2\ldots\rho_{d-2}}$ é um circuito de profundidade 2 e pelo Lema de Håstad com probabilidade pelo menos $1 - \frac{1}{2^{10}}$ colapsa para um árvode de decisão de profundidade 10 quando aplicamos $\rho \leftarrow R_{p_2}$ às variáveis que sobreviveram todas as restrições anteriores.

Primeiro nós temos que ao aplicar $\rho \leftarrow R_{p_1}$ às variáves de entrada de C, cada porta $\vee$ no segundo nível de C podem ser substituida por uma árvore de decisão de profundidade $\log(120S)$ com probabilidade pelo menos $2^{-\log(120S)} = 1/120S$ (isto é apenas uma aplicação do Lema de Håstad). Portanto, com probabilidade pelo menos $1 - S_2/120S$, onde em geral nós denotaremos por S_i o número de portas lógicas no i-ésimo nível de C, todas as portas $\vee$ no segundo nível de C podem ser substituidas por árvores de decisão de profundidade $\log(120S)$. Agora nós usamos que uma árvore de decisão de profundidade $\log(120S)$ pode ser representável por uma fórmula FNC de largura $\log(120S)$ para colapsar o segundo e terceiro nível de C obtendo então um circuito de profundidade $d-1$ e fan-in $\log(120S)$ no seu nível mais baixo.

Nós agora repetimos o mesmo processo $d-2$ vezes usando restrições $\rho_i \leftarrow R_{p_2}$, em cada passo reduzindo a profundidade do circuito $C_{|\rho_1\rho_2\ldots\rho_{i-1}}$ em um com probabilidade pelo menos $1 - S_1/120S$. No último passo, assumindo que $C_{|\rho_1\rho_2\ldots\rho_{d-2}}$ tenha com sucesso reduzido a um circuito de profundidade 2, ao aplicarmos $\rho_{d-2} \leftarrow R_{p_2}$ temos que com probabilidade pelo menos $1 - 2^{-10}$ obtemos uma árvore de decisão de profundidade no

máximo 10. A probabilidade que todas as restrições ρ_i tenha sucedidas em reduzir a profundidade de C é pelo menos

$$1 - S_2/120S - S_3/120S - \cdots - S_{d-2}/120S - 2^{-10} \geq 1 - 1/120 - 2^{-10} \geq 0,99.$$

Provando (2):

Nós notamos que $D(\text{Parity}_{n|\rho}) > 10$ sempre que o número de variáveis que continuam livres na restrição $\rho \leftarrow R_p$ for maior do que 10. Seja $\text{free}(\rho) = |\{i|\rho(i) = *\}|$, então pela desigualdade de Chernoff nós temos que

$$\Pr[\text{free}(\rho) \leq 10] \leq \exp\left(- \frac{n}{10w}\left(\frac{1}{10\log(120S)}\right)^{d-2}\frac{\delta^2}{2}\right),$$

onde $\delta = 1 - \frac{100w}{n}(10\log(120S))^{d-2}$. Nós queremos mostrar que a probabilidade acima é no máximo uma constante então é suficiente mostrar que o valor no expoente é $-\omega(1)$. Como $(1 - \frac{100w}{n}(10\log(120S))^{d-2})^2 \geq 1 - \frac{200w}{n}(10\log(120S))^{d-2}$, segue que

$$\begin{aligned}\frac{n}{10w}\left(\frac{1}{10\log(120S)}\right)^{d-2}\frac{\delta^2}{2} &\geq \frac{n}{10w}\left(\frac{1}{10\log(120S)}\right)^{d-2}\left(1/2 - \frac{100w}{n}(10\log(120S))^{d-2}\right)\\ &= \frac{n}{20w}\left(\frac{1}{10\log(120S)}\right)^{d-2} - 10.\end{aligned}$$

Lembrando que $S = 2^{10n^{\frac{1}{d-1}}}$ e $w = \log^c n$ temos que

$$\frac{n}{20w}\left(\frac{1}{10\log(120S)}\right)^{d-2} - 10 = \Omega\left(\frac{n^{1-\frac{d-2}{d-1}}}{\log^c n}\right)$$

o que é $\omega(1)$ e portanto para n suficientemente grande temos que isso é menor do que $0,01$, e portanto:

$$\Pr[\text{free}(\rho) > 10] \geq 0,99,$$

quando n é suficientemente grande.

□

Lembrando no capítulo anterior quando mostramos que $\mathsf{PH}^A \neq \mathsf{PSPACE}^A$ com probabilidade 1 para um oráculo aleatório (ver 3.10) nós precisamos do teorema 3.9 que diz que a função paridade não pode nem mesmo ser aproximada por circuitos de profundidade constante e tamanho $2^{o(n^{\frac{1}{d-1}})}$ (ou seja, o mesmo tipo de circuitos que consideramos no teorema 3.8). Nós na verdade podemos provar o teorema 3.9 a partir da prova do teorema 3.8 acima, obervando o seguinte:

Fato 4.13. *Seja $n \geq 1$ e T uma árvore de decisão de profundidade $\leq n - 1$, então*

$$\Pr[T(x) = \mathit{Parity}_n(x)] = 1/2.$$

Isto é verdade pois se considerarmos um caminho π de T e o conjunto $X_\pi = \{x \in \{0,1\}^n | x \text{ segue o caminho } \pi \text{ em } T\}$ então exatamente metade das strings $x \in X_\pi$ têm paridade igual ao valor da folha de π. Além disso nós também temos o seguinte fato:

Fato 4.14. *Seja $p \in (0,1)$ e considere a seguinte distribuição $\mathcal{D}$ de strings em $\{0,1\}^n$:*

1. *Tire uma restrição $\rho \leftarrow R_p$;*
2. *Tire uma string x' uniformemente de $\{0,1\}^{\rho^{-1}(*)}$.*

Então $\mathcal{D}$ é a distribuição uniforme sobre as strings em $\{0,1\}^n$.

A partir de 4.13 e 4.14 e pela prova do teorema 3.8 é verdade que qualquer vantagem que um circuito C que consideramos no teorema 3.8 sobre uma das funções constantes em aproximar a função Parity_n vem das restrições ρ tais que pelo menos uma das condições (1) e (2) na prova daquele teorema não é satisfeita. O que nós mostramos é exatamente que apenas uma fração pequena das restrições falham em satisfazer ambas as condições e portanto qualquer vantagem que C venha a ter é limitada por este fato. Nós temos que

$$\left| \Pr[C(x) = \text{Parity}_n(x)] - 1/2 \right| \leq 0,2.$$

Isso é uma asserção mais fraca do que foi citado no enunciado do teorema 3.9.

Circuitos de profundidade d para Parity_n

Nós acabamos de ver limitante inferior para o tamanho de um circuito de profundidade d para Parity_n quando d é constante. Então podemos nos perguntar quanto portas lógicas são o suficiente para um circuito de profundidade d poder computar Parity_n. Na verdade, nós podemos ver que o limitante inferior do teorema 3.8 é essencialmente ótimo como demostra o teorema a seguir.

Teorema 4.15. *Seja $d \geq 2$ uma constante. Então existe um circuito de profundidade d e tamanho $\mathcal{O}\left(n^{\frac{d-2}{d-1}} 2^{n^{\frac{1}{d-1}}}\right)$ com fan-in no nível mais baixo igual a $n^{\frac{d-2}{d-1}}$ que computa Parity_n.*

Demonstração. Seja $S^n(d)$ o tamanho do menor circuito de profundidade d que computa Parity_n. Nós provaremos por indução em d que

$$S^n(d) \leq 2^{n^{\frac{1}{d-1}}} \sum_{k=0}^{d-2} 2^{k-1} n^{\frac{k}{d-1}} + 1 = \mathcal{O}\left(n^{\frac{d-2}{d-1}} 2^{n^{\frac{1}{d-1}}}\right).$$

A última desigualdade é verdade pois o somatório acima é no máximo $d2^d n^{\frac{d-2}{d-1}}$ que por sua vez é $\mathcal{O}\left(n^{\frac{d-2}{d-1}}\right)$ (nós estamos considerando d constante).

Para $d = 2$, a fórmula FND que computa a soma dos mintermos da função Parity_n tem tamanho $2^{n-1} + 1$ e portanto o caso base é verdadeiro.

Agora seja $d > 2$. Nós podemos construir um circuito para Parity_n de profundidade d com o seguinte procedimento.

1. Particione as n variáveis de entrada em $n^{\frac{1}{d-1}}$ blocos de tamanho $m = n^{\frac{d-2}{d-1}}$ cada.
2. Use o circuito de profundidade $d-1$ e tamanho $S^m(d-1)$ para computar a paridade e a negação da paridade das variáveis de cada bloco.
3. Compute a paridade das saídas dos subcircuitos que computam Parity_m do passo (2) usando uma fórmula FND com $2^{n^{\frac{1}{d-1}}-1} + 1$ portas lógicas.

Agora nós analizamos quantas portas lógica o procedimento acima cria. Nós podemos ver que a profundidade do circuito é d já que podemos colapsar a porta de saída de cada subcircuito para Parity_m ou a sua negação com as portas $\wedge$ da fórmula FND que computa a paridade dos subcircuitos. Como temos $2n^{\frac{1}{d-1}}$ subcircuitos de tamanho $S^m(d-1)$ mais a fórmula FND que computa a paridade das saídas dos subcircuitos temos que o tamanho do circuito gerado é

$$2n^{\frac{1}{d-1}}(S^m(d-1)-1)+2^{n^{\frac{1}{d-1}}-1}+1.$$

Nós temos um fator $S^m(d-1)-1$ pois nós removemos uma porta lógica de cada subcircuito quando colapsamos um dos níveis do circuito. Pela hípotese da indução e por termos definido $S^n(d)$ como sendo o menor circuito que computa Parity_n:

$$S^n(d) \leq 2n^{\frac{1}{d-1}}2^{m^{\frac{1}{d-2}}}\sum_{k=0}^{d-3}2^{k-1}m^{\frac{k}{d-2}}+2^{n^{\frac{1}{d-1}}-1}+1.$$

E como $m = n^{\frac{d-2}{d-1}}$:

$$\begin{aligned}
S^n(d) &\leq 2n^{\frac{1}{d-1}}2^{n^{\frac{1}{d-1}}}\sum_{k=0}^{d-3}2^{k-1}n^{\frac{k}{d-1}}+2^{n^{\frac{1}{d-1}}-1}+1 \\
&= 2^{n^{\frac{1}{d-1}}}\sum_{k=1}^{d-2}2^{k-1}n^{\frac{k}{d-1}}+2^{n^{\frac{1}{d-1}}-1}+1 \\
&= 2^{n^{\frac{1}{d-1}}}\left(\sum_{k=1}^{d-2}2^{k-1}n^{\frac{k}{d-1}}+1/2\right)+1 \\
&= 2^{n^{\frac{1}{d-1}}}\sum_{k=0}^{d-2}2^{k-1}n^{\frac{k}{d-1}}+1.
\end{aligned}$$

O que conclui o último passo da indução.

Quanto ao fan-in do circuito, é suficiente notar que o fan-in do circuito é igual ao fan-in dos subcircuitos que computam a paridade das variáveis em cada bloco. Como cada bloco tem $n^{\frac{d-2}{d-1}}$ variáveis temos que o fan-in de cada subcircuito é no máximo este valor.

□

4.2 Projeções aleatórias e a prova de RST dos teoremas 3.12 e 3.14

Agora nós iremos ver como provar os teoremas 3.12 usando uma generalização de restrições aleatórias. Lembrando que quando o nosso objetivo era provar que a hierarquia polinomial é infinita relativa a um oráculo nós precisamos de uma hierarquia de circuitos de profundidade constante. Nós podemos nos perguntar se o método de restrições aleatórias que acabamos de ver é suficiente para provar que tal hierarquia existe. Infelizmente, este não é o caso por causa de uma diferença crucial entre provar que a função paridade não tem circuitos de profundidade constante e provar que existe uma hierarquia de circuitos de profundidade constante. O argumento clássico é que agora estamos querendo mostar que circuitos de profundidade d, para algum $d > 1$, são capazes de computar funções com uma quantidade polinomial de portas lógicas que circuitos de profundidade

$d-1$ não conseguem. Porém, o método de restrições aleatórias foi usada exatamente para "destruir" circuitos de profundidade constante, então não temos mais a capacidade de distinguir circuitos de profundidade $d-1$ da nossa função alvo por ela também se tratar de uma função com profundidade constante.

Para provar a existência de uma hierarquia de profundidade constante nós usamos as funções de Sipser que por conveniência definimos mais uma vez abaixo.

Definição 4.16. *(As funções de Sipser)*

Para $d \geq 2$ a função de Sipser $f^{m,d}$ é uma fórmula monotônica e read-once *onde o nível mais baixo tem fan-in m, as portas lógicas nos níveis 2 até d - 1 têm fan-in $w = 2^m mln(2)$ e a porta lógica no nível mais alto tem fan-in w_d que nós definiremos mais tarde. Ou seja, podemos escrever $f^{m,d}$ como*

$$\bigwedge_{i_1=1}^{m} \bigvee_{i_2=1}^{w} \cdots \bigvee_{i_{d-1}=1}^{w} \bigwedge_{i_d=1}^{w_d} x_{i_1,i_2,\ldots,i_d}, \text{ se } d \text{ é par.} \tag{4.7}$$

e

$$\bigwedge_{i_1=1}^{m} \bigvee_{i_2=1}^{w} \cdots \bigwedge_{i_{d-1}=1}^{w} \bigvee_{i_d=1}^{w_d} x_{i_1,i_2,\ldots,i_d}, \text{ se } d \text{ é ímpar.} \tag{4.8}$$

Nós então temos que definir uma nova forma de simplificar circuitos AC^0 que colapsa circuitos de profundidade $d-1$ ao mesmo tempo que mantém algum tipo de estrutura ao ser aplicada sobre a função $f^{m,d}$. Com este objetivo em mente define-se projeções aleatórias que são uma generalização de restrições.

Definição 4.17. *(Projeções aleatórias)*

Sejam $\mathcal{X}, \mathcal{Y}$ espaços de variáveis tais que $n = |\mathcal{X}| \leq |\mathcal{Y}|$ e seja block : $\mathcal{X} \to \mathcal{Y}$ uma função. Nós definimos um espaço de projeções aleatórias P a partir de um espaço de restrições R tal que $\rho \leftarrow P$ é formada da seguinte forma:

Seja $\rho' \in \{*, 0, 1\}^n$ uma restrição tirada de R, então

$$\rho(x) = \begin{cases} \rho'(x) & \text{se } \rho'(x) \in \{0,1\} \\ block(x) & \text{se } \rho'(x) = *. \end{cases}$$

Para cada variável $y \in \mathcal{Y}$, o conjunto $\{x \in \mathcal{X} | block(x) = y\}$ é denoninado o bloco de y. Se x pertence ao bloco de y e $\rho(x) = y$ nós dizemos que x foi projetada para y.

Nós podemos dizer que projeções são uma generalização de restrições pois uma restrição é uma projeção onde $\mathcal{X} = \mathcal{Y}$ e block é a função identidade. Durante esta subseção ao aplicarmos uma projeção ρ sobre as variáveis de uma função nós passamos a considerar a projeção de f com respeito a ρ que nós definimos a seguir.

Definição 4.18. *(Projeção de uma função)*

Seja $f : \{0,1\}^n \to \{0,1\}$ e P um espaço de projeção que mapeia as variáveis de entrada da função f para $\{0,1\} \cup \mathcal{Y}$. Seja $\rho \leftarrow P$, então a projeção de f com respeito a ρ é

$$Proj_\rho f : \mathcal{Y} \to \{0,1\}$$
$$Proj_\rho f(y_1, y_2, \ldots, y_m) \mapsto f(\rho(x_1), \rho(x_2), \ldots, \rho(x_n)),$$

onde $m = |\mathcal{Y}|$ e neste caso específico deve-se entender $\rho(x_i)$ como carregando o mesmo valor que y_j sempre que $\rho(x_i) = y_j$ (ao invés de ver $\rho(x_i)$ como uma variável formal em $\mathcal{Y}$ sem nenhum valor atribuido como na definição 4.17).

Projeções aleatórias vão ter o mesmo papel na prova dos teoremas 3.12 e 3.14 que restrições aleatórias tiveram nas provas dos teoremas 3.8 e 3.9. Para provar que $f^{m,d}$ não pode ser computada por circuitos de profundidade $d-1$ e tamanho subexponencial nós iremos usar uma sequência de espaços de projeções P_i, para $i = 1, 2, \ldots, d-1$, que irão satisfazer as três condições listadas a seguir:

1. Qualquer circuito C de profundidade $d-1$ e tamanho subexponencial colapsa para uma função simples quando aplicamos $\rho_1, \rho_2, \ldots \rho_{d-1}$ às variáveis de entrada de C, onde cada ρ_i é tirada de P_i.
2. A função $f^{m,d}$ mantém estrutura ao aplicarmos $\rho_1, \rho_2, \ldots, \rho_{d-1}$ às suas variáveis.
3. As projeções $\rho_1, \rho_2, \ldots, \rho_{d-1}$ completam para a distribuição uniforme sobre $\{0,1\}^n$.

O item (3) é necessário para a prova do teorema 3.14 e ela tem o mesmo papel que 4.14 tem para restrições aleatórias. Um ponto importante é que a definição de cada um dos espaços de projeções P_i dependem das projeções anteriores. Nós iremos definir $P_i(\rho_{i-1})$, onde $\rho_{i-1} \leftarrow P_{i-1}$, mas na maioria da vez nós iremos deixar implícita a dependência sobre ρ_{i-1}.

Nós definimos cada espaço de projeção P_i com a função $f^{m,d}$ em mente. Sejam $A_0, A_1, \ldots, A_d$ espaços de variáves em que A_0 é o conjunto das variáveis de entrada de $f^{m,d}$ e para cada A_i, com $i > 0$, A_i tem uma variável x_a para cada porta lógica a que aparece no i-ésimo nível da fórmula $f^{m,d}$. Nós faremos um abuso de notação e escreveremos $a \in A_i$ para denotar que a é uma porta lógica no i-ésimo nível de $f^{m,d}$. Para $a \in A_i$ seja $\text{input}(a)$ o conjunto de portas lógicas ou variáveis de entrada que alimentam a. Com isto dito, o espaço de projeção P_i irá mapear variáveis em A_{i-1} para $\{0,1\} \cup A_i$ em que para cada $x_a \in A_i$ é verdade que uma variável $x_{a'} \in A_{i-1}$ pode ser projetada para x_a se e somente se $a' \in \text{input}(a)$, em outras palavras $x_{a'}$ pertence ao bloco de x_a se e somente se $a' \in \text{input}(a)$.

O espaço de projeções P_1

Vamos começar definindo P_1 e depois mostramos como cada P_i, para $i > 1$, é definida a partir da projeção ρ_{i-1} tirada de P_{i-1}. Ao definir cada P_i nós iremos atribuir um valor $\rho_i(x_a)$ para cada $x_a \in A_i$ que denotará o valor na saída da porta lógica a após aplicarmos a projeção aleatória. Apesar disso não aparecer na definição 4.17 de projeções aleatórias, é útil para nós guardar este valor.

Durante esta seção nós iremos considerar os seguintes parâmetros:

$$\lambda = 2^{-5m/4} \qquad q = 2^{-m/2} - 2^{-10m/9}. \tag{4.9}$$

Definição 4.19. *(O espaço de projeção P_1)*

Por conveniência iremos considerar apenas as variáveis em uma porta $a \in A_1$ específica pois as projeções $\rho_1 \leftarrow P_1$ atribuem valores às variáveis que alimentam portas diferentes de forma independente. Inicialmente nós definimos o valor $\rho_1(x_a)$ da seguinte forma:

$$\rho_1(x_a) = \begin{cases} 1 & \text{com probabilidade } \lambda. \\ x_a & \text{com probabilidade } q. \\ 0 & \text{com probabilidade } 1-\lambda-q. \end{cases} \tag{4.10}$$

Em seguida nós tiramos um subconjunto não vazio S de input(a) aleatoriamante e uniformemente e fazemos $\rho_1(x_{a'}) = 1$ para todo $x_{a'} \in \text{input}(a) \setminus S$ e $\rho_1(x_{a'}) = \rho_1(x_a)$ para todo $x_{a'} \in S$.

Nós fazemos a seguinte observação importante. Seja $a \in A_1$ e vamos considerar uma string $\mathbf{x} \in \{0,1\}^m$ formada pelo seguite procedimento. Seja t_1 tal que

$$\lambda + qt_1 = 2^{-m}. \tag{4.11}$$

Ou seja,

$$t_1 = \frac{2^{-m} - \lambda}{q} = \frac{2^{-m} - 2^{-5m/4}}{2^{-m/2} - 2^{-10m/9}}, \tag{4.12}$$

o que é bem próximo de $2^{-m/2}$. Então prosseguimos da seguinte forma:

1. Se $\rho_1(x_a) \in \{0,1\}$:
   Faça $\mathbf{x}_{a'} = \rho_1(x_{a'})$ para todo $a' \in \text{input}(a)$.
2. Se $\rho_1(x_a) = x_a$:
   Faça $\mathbf{x}_{a'} = 1$ para todo $a' \in \text{input}(a)$ tal que $\rho_1(x_{a'}) = 1$ e $\mathbf{x}_{a'} = b$ para todos os outros $\mathbf{x}_{a'} \in \text{input}(a)$, em que $b \sim \{0_{1-t_1}, 1_{t_1}\}$.

Nós temos que a distribuição acima é equivalente à distribuição uniforme sobre strings em $\{0,1\}^m$. Para ver isto nós mostramos que a probabilidade que a string 1^m é gerada pelo procedimento acima é 2^{-m} (isso é suficiente para provar que a distribuição de strings geradas é uniforme pois podemos facilmente observar que todas as outras strings são geradas com a mesma probabilidade). Isto segue direto da nossa definição de λ, q e t_1 em 4.9, 4.11 e 4.12, e pela primeira e segunda linha de 4.10:

$$\Pr[\mathbf{x} = 1^m] = \lambda + qt_1 = 2^{-m}$$

Ou seja, $\mathbf{x}$ é 1^m sempre que $\rho_1(x_a) = 1$ ou $\rho_1(x_a) = x_a$ e fazemos $\mathbf{x}_{a'} = 1$ para todos $x_{a'} \in \text{input}(a)$ no passo (2) do nosso procedimento. Aplicando o mesmo procedimento para todas $a \in A_1$ nós obtemos a distribuição uniforme sobre $\{0,1\}^n$.

O que o procedimento acima faz é basicamente atribuir valores tirados aleatoriamente de $\{0_{1-t_i}, 1_{t_i}\}$ às variáveis em A_1 que sobreviveram ρ_1. Então, se temos uma projeção $\rho_1 \leftarrow P_1$ e para cada $a \in A_1$ tal que $\rho_1(x_a) = x_a$ nós fizermos $x_a \sim \{0_{1-t_1}, 1_{t_1}\}$, nós temos uma distribuição de valores para as variáveis em A_1 que é equivalente a se nós tivessemos tirado uma atribuição às variáveis de entrada de $f^{m,d}$ da distribuição uniforme e olhassemos para a saída de cada porta $\wedge$ no primeiro nível da fórmula $f^{m,d}$.

Para $i > 1$ defina recursivamente

$$t_i = \frac{(1 - t_{i-1})^{qw} - \lambda}{q}. \tag{4.13}$$

A idéia agora é que para todos espaços de projeções P_i subsequentes nós iremos garantir que ao substituir as variáveis que sobreviveram a projeção por um valor aleatório tirado da distribuição $\{0_{1-t_i}, 1_{t_i}\}$ se i for ímpar ou $\{0_{t_i}, 1_{1-t_i}\}$ se i for par, nós também iremos obter a distribuição que teriamos se nós tivessemos tirado uma atribuição às variáveis de entrada da distribuição uniforme e olhassemos para o valor na saída de cada porta lógica em A_i (ou seja, P_i completa para a distribuição uniforme). Além disso, nós iremos usar o fato que P_{i-1} completa para a distribuição uniforme para provar o mesmo para P_i. Assim, para provar que um circuito C de tamanho subexponencial e profundidade $d-1$ não é nem mesmo correlacionado com $f^{m,d}$ nós iremos aplicar uma sequência de projeções $\rho_1\rho_2 \ldots \rho_{d-1}$, onde cada ρ_i é tirada de P_i, ao circuito C e à $f^{m,d}$ e usando a notação $\Psi = \text{Proj}_{\rho_1\rho_2\ldots\rho_{d-1}}$ teremos que:

$$E_{\Psi}\left[\Pr_{\mathbf{x}\sim\{0_{1-t_{d-1}},1_{t_{d-1}}\}^{A_{d-1}}}\left[\Psi(f^{m,d})(\mathbf{x})\neq\Psi(C)(\mathbf{x})\right]\right]=\Pr_{\mathbf{x}\sim\{0_{1/2},1_{1/2}\}^{n}}\left[f^{m,d}(\mathbf{x})\neq C(\mathbf{x})\right],\text{ se d for par,}\quad(4.14)$$

ou

$$E_{\Psi}\left[\Pr_{\mathbf{x}\sim\{0_{t_{d-1}},1_{1-t_{d-1}}\}^{A_{d-1}}}\left[\Psi(f^{m,d})(\mathbf{x})\neq\Psi(C)(\mathbf{x})\right]\right]=\Pr_{\mathbf{x}\sim\{0_{1/2},1_{1/2}\}^{n}}\left[f^{m,d}(\mathbf{x})\neq C(\mathbf{x})\right],\text{ se d for ímpar,}\quad(4.15)$$

Então nós mostraremos que $\Psi(f^{m,d})$ e $\Psi(C)$ em si não são correlacionadas para mostrar que $f^{m,d}$ e C também não são correlacionadas.

A observação acima corresponde à terceira condição que os espaços de projeção devem satisfazer. A primeira condição (que os circuitos C simplificam quando aplicamos $\Psi=\mathrm{Proj}_{\rho_1\rho_2\ldots\rho_{d-1}}$) nós iremos ver mais pra frente quando provarmos um lema da troca para os espaços de projeções P_i. Para a segunda condição ($f^{m,d}$ mantém estrutura) nós iremos tratar agora o caso i = 1 fazendo as seguintes observações.

Definição 4.20. *(Projeções típicas para P_1)*

Seja $\rho_1 \leftarrow P_1$. Para toda $a' \in A_2$ seja $S_{a'} = \{a \in input(a') | \rho_1(x_a) = x_a\}$ e para toda $a'' \in A_3$ seja $U_{a''} = \{a' \in input(a'') | x_{a'}$ não é feita constante pela projeção $\rho_1\}$. Nós dizemos que ρ_1 é típica se:

1. *Para todo $a' \in A_2$, $\left||S_{a'}| - qw\right| \leq w^{1/3}$.*
2. *Para todo $a'' \in A_3$, $|U_{a''}| \geq w - w^{4/5}$.*

Note que qualquer restrição $\rho_1 \leftarrow P_1$ típica não fixa nenhuma porta $\vee$ a' em A_2 com o valor zero, pois para isto nós teriamos que ter $\rho_1(x_a) = 0$ para todo $a \in input(a')$, o que contradiz a condição (1). Portanto podemos equivalentemente dizer que a condição (2) exige que não mais do que $w^{4/5}$ portas $\vee$ na entrada de $a'' \in A_3$ tenham sido fixadas com o valor 1 pela projeção ρ_1.

Nós iremos entender a idéia por trás da primeira condição que uma projeção típica tem que satisfazer depois de definirmos os espaços de projeções P_i para $i > 1$ e após vermos o lema da troca para projeções P_i. Nós precisamos da segunda condição porque queremos que ao aplicarmos ρ_1 às variáveis de entrada de $f^{m,d}$ o circuito por trás da função $f^{m,d}$ a partir do terceiro nível não seja afetado.

Então temos que dado que $\rho_1 \leftarrow P_1$ é típica então $\mathrm{Proj}_{\rho_1} f^{m,d}$ é uma fórmula com profundidade $d-1$ sobre as variáveis em A_1 que sobreviveram ρ_1 (ou seja, não foram feitas constantes). No nível mais baixo de $\mathrm{Proj}_{\rho_1} f^{m,d}$ temos portas $\vee$ com fan-in no intervalo $[qw - w^{1/3}, qw + w^{1/3}]$ e no segundo nível temos portas $\wedge$ com fan-in $\geq w - w^{4/5}$. O restante da fórmula permanece intacta. Para mostrar que $f^{m,d}$ mantém estrutura com alta probabilidade nós temos primeiro que argumentar que $\rho_1 \leftarrow P_1$ é típica com alta probabilidade.

Proposição 4.21. *Seja $\rho_1 \leftarrow P_1$, então:*

1. *Para todo $a' \in A_2$, $\Pr\left[\left||S_{a'}| - qw\right| \leq w^{1/3}\right] = 1 - e^{-\widetilde{\Omega}(w^{1/6})}$;*
2. *Para todo $a'' \in A_3$, $\Pr\left[|U_{a''}| \geq w - w^{4/5}\right] = 1 - e^{-\widetilde{\Omega}(w^{4/5})}$.*

Daí segue pelo Princípio da Inclusão-Exclusão que ρ_1 é típica com probabilidade $1 - \left(|A_2| + |A_3|\right)e^{-\widetilde{\Omega}(w^{1/6})}$. Como $|A_i| = w^{\mathcal{O}(d)}$, para todo $i \in [d]$, temos que ρ_1 é típica como probabilidade $1 - e^{-\widetilde{\Omega}(w^{1/6})}$.

Demonstração. Nós podemos provar ambos os itens no enunciado aplicando a desigualdade de Chernoff. Vale lembrar que se $a'' \in A_3$ e $a' \in \text{input}(a'')$ então $a' \in A_2$. Similarmente, se $a' \in A_2$ e $a \in \text{input}(a')$ então $a \in A_1$.

1. Provando o item (1).

   Seja $a' \in A_2$. Uma variável $x_a \in \text{input}(a')$ satisfaz $\rho_1(x_a) = x_a$ com probabilidade q pela definição de P_1 em 4.19. Portanto, temos que $E[|S_{a'}|] = qw = \widetilde{\Theta}(w^{1/2})$, e pela desigualdade de Chernoff temos que

   $$\Pr\left[\big||S_{a'}| - qw\big| \geq w^{1/3}\right] \leq e^{-\frac{\delta^2}{2+\delta}qw},$$

   onde δ satisfaz $\delta qw = w^{1/3}$. Equivalentemente, $\delta = w^{1/3}/qw$, então segue que $\delta = \widetilde{\Omega}(w^{-1/6})$, onde nós também usamos que $qw = \widetilde{\Theta}(w^{1/2})$, e portanto

   $$e^{-\frac{\delta^2}{2+\delta}qw} = e^{-\widetilde{\Omega}(w^{1/6})}.$$

   E por fim podemos concluir que

   $$\Pr\left[\big||S_{a'}| - qw\big| \leq w^{1/3}\right] = 1 - e^{-\widetilde{\Omega}(w^{1/6})}.$$

2. Provando o item (2).

   Seja $a'' \in A_3$ e $a' \in \text{input}(a'')$. Como nós observamos em 4.20 nós podemos assumir que a' é substituida pela constante 1 sempre que $a' \notin U_{a''}$. Temos que a' não é forçada para 1 por ρ_1 se todos $a \in \text{input}(a')$ satisfazem $\rho_1(x_a) \neq 1$ (pois a' é uma porta $\vee$), o que acontece com probabilidade $(1-\lambda)^w$ pela definição de P_1 em 4.19 e por ρ_1 agir de forma independente nos blocos $a \in A_1$. Segue então que $E\big[|U_{a''}|\big] = w(1-\lambda)^w$. Como $(1-\lambda)^w \geq 1 - w\lambda$ também temos que

   $$E\big[|U_{a''}|\big] = w(1-\lambda)^w \geq w(1-w\lambda),$$

   e pela desigualdade de Chernoff segue que

   $$\Pr\left[w - |U_{a''}| \geq w^{4/5}\right] \leq e^{-\frac{\delta^2}{2+\delta}(w - E[|U_{a''}|])},$$

   onde δ é tal que $(1+\delta)(w - E[|U_{a''}|]) = w^{4/5}$. Como $w - E\big[|U_{a''}|\big] \leq w(1-(1-w\lambda)) = w^2\lambda$ podemos dizer que $\delta \geq \frac{w^{4/5}}{w^2\lambda} - 1 = \widetilde{\Omega}(w^{1/20})$. Portanto,

   $$e^{-\frac{\delta^2}{2+\delta}(w - E[|U_{a''}|])} = e^{-\widetilde{\Omega}(w^{1/20})(w - E[|U_{a''}|])}.$$

   Como $(1-\lambda)^w \leq e^{-w\lambda} \leq 1 - w\lambda + \frac{w^2\lambda^2}{2}$, temos que $w^2\lambda - \frac{w^3\lambda^2}{2} \leq w - E\big[|U_{a''}|\big] \leq w^2\lambda$. Ou seja, $w - E\big[|U_{a''}|\big] = \widetilde{\Theta}(w^{3/4})$. Então:

   $$e^{-\widetilde{\Omega}(w^{1/20})(w - E[|U_{a''}|])} = e^{-\widetilde{\Omega}(w^{4/5})}.$$

   Daí podemos concluir que

   $$\Pr\left[w - |U_{a''}| \leq w^{4/5}\right] = 1 - e^{-\widetilde{\Omega}(w^{4/5})},$$

   o que é equivalente à

   $$\Pr\left[|U_{a''}| \geq w - w^{4/5}\right] = 1 - e^{-\widetilde{\Omega}(w^{4/5})}.$$

□

Os espaços de projeções P_i

Agora nós definimos P_i para cada $i > 1$ de forma análoga a como definimos P_1. Em seguida nós também iremos ver que cada P_i completa para a distribuição uniforme e estendemos a definição de projeções típicas para P_i e mostraremos que $\rho_i \leftarrow P_i$ é típica com alta probabilidade.

A partir de agora nós iremos usar a função g definida como:

$$g(i,d) = 1/3 + \frac{i-1}{12d}.$$

Vale notar que para $1 \leq i \leq d-1$, temos $1/3 \leq g(i,d) \leq 5/12 < 1/2$.

Definição 4.22. *(O espaço de projeção P_i)*

De novo, nós nos concentramos em uma única porta $a \in A_i$ e sem perda de generalidade nós assumimos que i é ímpar (e portanto a é uma porta $\wedge$). Nós não perdemos em generalidade pois o caso em que i é par é completamente análogo com os papeis de 0 e 1 trocados. Seja $S_a = \{x_{a'} \in input(a) | \rho_{i-1}(x_{a'}) = x_{a'}\}$. Primeiro nós iremos "rejeitar" ρ_{i-1} se $\rho_{i-1}(x_{a'}) = 0$ para algum $x_{a'} \in input(a)$ ou se $\big||S_a| - qw\big| > w^{g(i-1,d)}$. Ao rejeitar ρ_{i-1} nós fazemos $x_{a'} \sim \{0_{t_{i-1}}, 1_{1-t_{i-1}}\}$ para cada $x_{a'} \in S_a$. Suponha que ρ_{i-1} não foi rejeitada, então nós fazemos

$$\rho_i(x_a) = \begin{cases} 1 & \text{com probabilidade } \lambda. \\ x_a & \text{com probabilidade } q_a. \\ 0 & \text{com probabilidade } 1 - \lambda - q_a. \end{cases}$$

Nós escolhemos q_a de forma que $\lambda + q_a t_i = (1-t_{i-1})^{|S_a|}$ (ou seja, $q_a = \frac{(1-t_{i-1})^{|S_a|} - \lambda)}{t_i}$). Em seguida tiramos um subconjunto não vazio T de S_a onde cada variável em S_a é incluida em T com probabilidade t_{i-1} e fazemos $\rho_i(x_{a'}) = 1$ para todo $x_{a'} \in S_a \setminus T$ e $\rho_i(x_{a'}) = \rho_i(x_a)$ para todo $x_{a'} \in T$.

Daqui pra frente a seguinte relação entre q e q_a, para qualquer $a \in \bigcup_{i=2}^{d-1} A_i$, nos será útil.

Proposição 4.23. *Seja $a \in A_i$ e $q_a = \frac{(1-t_{i-1})^{|S_a|} - \lambda}{t_i}$, onde $\big||S_a| - qw\big| \leq w^{g(i-1,d)}$, então*

$$q(1 - 3w^{g(i-1,d)} t_{i-1}) \leq q_a \leq q(1 + 3w^{g(i-1,d)} t_{i-1}).$$

Mais pra frente nós será útil a seguinte relação mais fraca entre q e q_a:

$$q/2 \leq q_a \leq 2q. \tag{4.16}$$

Agora iremos usar a hipótese que P_{i-1} completa para a distribuição uniforme para provar que P_i também completa para a distribuição unifome. Caso ρ_{i-1} não tenha sido rejeitada nós aplicamos o seguinte procedimento análogo ao que vimos após a definição de P_1. Seja $a \in A_i$:

1. Se $\rho_i(x_a) \in \{0,1\}$:
   Faça $\mathbf{x}_{a'} = \rho_i(x_{a'})$ para todo $a' \in \text{input}(a)$.

2. Se $\rho_i(x_a) = x_a$:

   Faça $\mathbf{x}_{a'} = 1$ para todo $a' \in \text{input}(a)$ tal que $\rho_i(x_{a'}) = 1$ e para todos os outros a' faça $\mathbf{x}_{a'} = b$ onde

$$b = \begin{cases} 0 & \text{com probabilidade } 1 - t_i \\ 1 & \text{com probabilidade } t_i. \end{cases}$$

Desta forma, para cada a' tal que $x_{a'} \in S_a$, nós temos o seguinte:

$$\Pr[\mathbf{x}_{a'} = 1] = 1 - \frac{t_{i-1}}{1-(1-t_{i-1})^{|S_a|}} + \frac{t_{i-1}}{1-(1-t_{i-1})^{|S_a|}}\big(\lambda + q_a t_i\big).$$

Lembrando que $\lambda + q_a t_i = (1-t_{i-1})^{|S_a|}$ pela forma como definimos q_a:

$$\begin{aligned} \Pr[\mathbf{x}_{a'} = 1] &= 1 - \frac{t_{i-1}}{1-(1-t_{i-1})^{|S_a|}} + \frac{t_{i-1}(1-t_{i-1})^{|S_a|}}{1-(1-t_{i-1})^{|S_a|}} \\ &= 1 - \frac{t_{i-1}}{1-(1-t_{i-1})^{|S_a|}}\big(1-(1-t_{i-1})^{|S_a|}\big) \\ &= 1 - t_{i-1}. \end{aligned}$$

Ou seja, atribuir um valor tirado da distribuição $\{0_{1-t_i}, 1_{t_i}\}$ às variáveis que sobreviveram ρ_i é equivalente a ter setado todas as variáveis em S_a em um valor tirado de $\{0_{t_{i-1}}, 1_{1-t_{i-1}}\}$. Como $i-1$ é par e pela hipótese que P_{i-1} completa para a distribuição uniforme devemos ter que $\rho_i \leftarrow P_i$ também completa para a distribuição uniforme. No caso em que ρ_{i-1} é rejeitada nós temos de imediato que as variáveis em S_a são atribuidas com valores tirados de $\{0_{t_{i-1}}, 1_{1-t_{i-1}}\}$ e portanto ao todo P_i completa para a distribuição uniforme. Nós temos a seguinte proposição como consequência.

Proposição 4.24. *Levando em conta a notação $\Psi = Proj_{\rho_1\rho_2\ldots\rho_{d-1}}$, onde cada projeção ρ_i foi tirada de P_i, então assumindo que d é par (o caso em que d é ímpar é simétrico) temos que para todas funções $f : \{0,1\}^n \to \{0,1\}$:*

$$E_\Psi\left[\Pr_{\boldsymbol{x}\sim\{0_{1-t_{d-1}},1_{t_{d-1}}\}^{A_{d-1}}}[\Psi(f)(\boldsymbol{x}) = 1]\right] = \Pr_{\boldsymbol{x}\sim\{0,1\}^n}[f(\boldsymbol{x}) = 1].$$

E então temos o seguinte corolário.

Corolário 4.25. *Para todas as funções $f, g : \{0,1\}^n \to \{0,1\}$ e assumindo que d é par (o caso em que d é ímpar é de novo simétrico) é verdade que*

$$E_\Psi\left[\Pr_{\boldsymbol{x}\sim\{0_{1-t_{d-1}},1_{t_{d-1}}\}^{A_{d-1}}}[\Psi(f)(\boldsymbol{x}) \neq \Psi(g)(\boldsymbol{x})]\right] = \Pr_{\boldsymbol{x}\sim\{0,1\}^n}[f(\boldsymbol{x}) \neq g(\boldsymbol{x})].$$

Em que $\Psi = Proj_{\rho_1\rho_2\ldots\rho_{d-1}}$.

Agora sabemos que para provar que um circuito arbitrário C de tamanho subexponencial e profundidade d - 1 não pode aproximar $f^{m,d}$ basta nós mostrarmos que com alta probabilidade $\Psi(C)$ e $\Psi(f^{m,d})$ são funções não correlacionadas.

Agora nos voltamos para o objetivo de garantir que os espaços de projeções $P_2, P_3, \ldots, P_{d-1}$ mantém a função $f^{m,d}$ bem estruturada. Nós prosseguimos de forma bem parecida de como fizemos para o espaço de projeção P_1. Primeiro nós iremos definir o que é uma projeção típica para P_i.

Definição 4.26. *(Projeções típicas para P_i)*

Para algum $i \in \{2,3,\ldots,d-1\}$ seja $\rho_i \leftarrow P_i$. Para toda variável $x_{a'} \in A_{i+1}$ seja $S_{a'} = \{x_a \in input(a') | \rho_i(x_a) = x_a\}$ e para todo variável $x_{a''} \in A_{i+2}$ seja $U_{a''} = \{x_{a'} \in input(a'') | x_{a'}$ não é feita constante pela projeção $\rho_i\}$. Nós dizemos que ρ_i é típica se:

1. *Para toda variável $x_a \in A_{i+1}$:*
   (a) $\big||S_a| - qw\big| \leq w^{g(i,d)}$, *se* $i < d-1$.
   (b) $\big||S_a| - qw_d\big| \leq w^{g(i,d)}$, *se* $i = d-1$
2. *Para todo variável $x_a \in A_{i+2}$:*
   (a) $|U_a| \geq w - w^{4/5}$, *se* $i < d-2$.
   (b) $|U_a| \geq w_d - w_d^{4/5}$, *se* $i = d-2$.

Se $i = d-1$ nós ignoramos o item (2).

Portanto dado que ρ_i é típica temos que $\mathrm{Proj}_{\rho_1\rho_2\ldots\rho_i} f^{m,d}$ tem a mesma estrutura que $\mathrm{Proj}_{\rho_1} f^{m,d}$ quando ρ_1 é típica mas com profundidade $d-i$ ao invés de $d-1$. Nós queremos provar que ρ_i é típica com alta probabilidade e para isso é suficiente provar que $\rho_i \leftarrow P_i(\rho_{i-1})$ é típica com alta probabilidade quando $\rho_{i-1} \leftarrow P_{i-1}$ por sua vez também é típica. Nós temos a seguite proposição análoga à 4.21.

Proposição 4.27. *Seja $\rho_{i-1} \leftarrow P_{i-1}$ uma projeção típica e $\rho_i \leftarrow P_i(\rho_{i-1})$, para algum $i \in \{2,3,\ldots,d-1\}$, então:*

1. *Para todo $a \in A_{i+1}$:*
   (a) $\Pr\Big[\big||S_a| - qw\big| \leq w^{g(i,d)}\Big] = 1 - e^{-\widetilde{\Omega}(w^{1/6})}$, *se* $i < d-1$.
   (b) $\Pr\Big[\big||S_a| - qw_d\big| \leq w^{g(i,d)}\Big] = 1 - e^{-\widetilde{\Omega}(w^{1/6})}$, *se* $i = d-1$.
2. *Para todo $a \in A_{i+2}$:*
   (a) $\Pr\big[|U_a| \geq w - w^{4/5}\big] = 1 - e^{-\widetilde{\Omega}(w^{4/5})}$, *se* $i < d-2$;
   (b) $\Pr\big[|U_a| \geq w_d - w_d^{4/5}\big] = 1 - e^{-\widetilde{\Omega}(w^{4/5})}$, *se* $i = d-2$.

Daí segue pelo Princípio da Inclusão-Exclusão que ρ_i é típica com probabilidade $1-\big(|A_{i+1}|+|A_{i+2}|\big)e^{-\widetilde{\Omega}(w^{1/6})} = 1 - e^{-\widetilde{\Omega}(w^{1/6})}$. *Se $i = d-1$ nós ignoramos o item (2).*

Demonstração. Nós de novo podemos usar a desigualdade de Chernoff para provar ambos os itens.

1. Provando o item (1).

   Primeiro vamos assumir que $i < d-1$. Seja $a' \in A_{i+1}$. Agora temos que $a \in \mathrm{input}(a')$ satisfaz $\rho_{i-1}(x_a) = x_a$ com probabilidade q_a e portanto $E[|S_{a'}|] = \sum_{a \in S_{a'}} q_a$. Usando a estimativa mais fraca para q_a em 4.16 e o fato que ρ_{i-1} é típica temos que $\frac{q}{2}(w - w^{4/5}) \leq E\big[|S_{a'}|\big] \leq 2qw$, ou seja $\mu = \widetilde{\Theta}(w^{1/2})$. Pela desigualdade de Chernoff temos que

$$\Pr\Big[\big||S_{a'}| - qw\big| \geq w^{g(i,d)}\Big] = e^{-\min\left(\frac{\delta_1^2}{2+\delta_1}, \frac{\delta_2^2}{2}\right)\times\widetilde{\Omega}(w^{1/2})}, \tag{4.17}$$

em que δ_1 satisfaz $(1+\delta_1)(qw+3qw^{g(i-1,d)+1}t_{i-1}) \geq qw+w^{g(i,d)}$ e δ_2 satisfaz $(1-\delta_2)(qw-3qw^{g(i-1,d)+1}t_{i-1}) \leq qw - w^{g(i,d)}$ (nós estamos usando 4.23). Então temos $\delta_1 = \widetilde{\Omega}(w^{g(i,d)-1/2})$ e $\delta_2 = \widetilde{\Omega}(w^{g(i,d)-1/2})$, e como $g(i,d) - 1/2 \geq -1/6$ nós obtemos o seguinte quando trocamos $\min\left(\frac{\delta_1^2}{2+\delta_1}, \frac{\delta_2^2}{2}\right)$ por um fator $\widetilde{\Omega}(w^{-1/6})$ no lado direito da desigualdade 4.17:

$$\Pr\Big[\big||S_{a'}| - qw\big| \geq w^{g(i,d)}\Big] = e^{-\widetilde{\Omega}(w^{1/6})}.$$

Daí temos que

$$\Pr\Big[\big||S_{a'}| - qw\big| \leq w^{g(i,d)}\Big] = 1 - e^{-\widetilde{\Omega}(w^{1/6})}.$$

Como os mesmos argumentos acima funcionam se trocarmos w por w_d nós também obtemos o resultado para o caso em que $i = d-1$.

2. Provando o item (2).

   Nós iremos provar o item (2) considerando separadamente o caso em que $i < d-2$ e o caso $i = d-2$, apesar de ambos os casos serem bastantes parecidos. Em ambos os casos nós iremos usar o fato que nenhuma porta $\vee$ no $(i+1)$-ésimo nível é fixada em 0 quando ρ_i é típica.

   (a) O caso $i < d-2$.

   Seja $a'' \in A_{i+2}$. Considere um $a' \in \text{input}(a'')$, por estarmos assumindo que i é ímpar temos que a' é uma porta $\vee$ e portanto ρ_i fixa a' em 1 se existe um $a \in \text{input}(a')$ tal que $\rho_i(x_a) = 1$. Para cada $a \in \text{input}(a')$ temos que $\Pr[\rho_i(x_a) = 1] = \lambda$ pois estamos assumindo que ρ_{i-1} é típica. Seja $\text{input}_{\rho_{i-1}}(a') = \{a \in \text{input}(a') | x_a \text{ não foi feita constante por } \rho_{i-1}\}$, então por ρ_{i-1} ser típica temos que $w - w^{4/5} \leq |\text{input}_{\rho_{i-1}}(a')| \leq w$. Daí temos que

   $$E\big[|U_{a''}|\big] = \sum_{a' \in \text{input}(a'')} (1-\lambda)^{|\text{input}_{\rho_{i-1}}(a')|}.$$

   e

   $$w(1-\lambda)^w \leq E\big[|U_{a''}|\big] \leq w(1-\lambda)^{w-w^{4/5}}.$$

   Usando que $w(1-\lambda)^{w-w^{4/5}} \leq we^{-(w-w^{4/5})\lambda} \leq w\big(1 - (w-w^{4/5})\lambda + \frac{(w-w^{4/5})^2\lambda^2}{2}\big)$ e $w(1-\lambda)^w \geq w(1-w\lambda)$, nós temos as seguintes desigualdades:

   $$w(w-w^{4/5})\lambda - \frac{w(w-w^{4/5})^2\lambda^2}{2} \leq w - E\big[|U_{a''}|\big] \leq w^2\lambda,$$

   e podemos concluir que $w - E\big[|U_{a''}|\big] = \widetilde{\Theta}(w^{3/4})$. Então, pela desigualdade de Chernoff temos que

   $$\Pr\Big[w - |U_{a''}| \geq w^{4/5}\Big] \leq e^{-\frac{\delta^2}{2+\delta}\widetilde{\Omega}(w^{3/4})},$$

   onde δ satisfaz $(1+\delta)w^2\lambda \geq w^{4/5}$, o que implica em $\delta = \widetilde{\Omega}(w^{1/20})$. Substituindo na desigualdade acima e rearranjando nós podemos concluir que

   $$\Pr\Big[|U_{a''}| \geq w - w^{4/5}\Big] \geq 1 - e^{-\widetilde{\Omega}(w^{4/5})}.$$

(b) O caso $i = d - 2$.
Seja $a'' \in A_d$ (como $|A_d| = 1$ temos que a'' é necessariamente a porta de saída da fórmula $f^{m,d}$). Nós prosseguimos da mesma maneira como na prova do caso $i < d - 2$ e obtemos as desigualdades

$$w_d(w - w^{4/5})\lambda - \frac{w_d(w - w^{4/5})^2\lambda^2}{2} \leq w_d - E\big[|U_{a''}|\big] \leq w_d w\lambda.$$

Mas como $w_d = \widetilde{\Theta}(w)$ nós ainda temos que $w_d - E\big[|U_{a''}|\big] = \widetilde{\Theta}(w^{3/4})$. Então pela desigualdade de Chernoff,

$$\Pr[w_d - |U_{a''}| \geq w_d^{4/5}] \leq e^{-\frac{\delta^2}{2+\delta}\widetilde{\Omega}(w^{3/4})},$$

em que δ satisfaz $(1+\delta)w_d w\lambda \geq w_d^{4/5}$, e portanto $\delta = \widetilde{\Omega}(w^{1/20})$. Por fim obtemos que

$$\Pr[|U_{a''}| \geq w_d - w_d^{4/5}] \geq 1 - e^{-\widetilde{\Omega}(w^{4/5})}.$$

□

Nós já podemos mostrar que a função $f^{m,d}$ mantém alguma estrutura quando aplicamos as projeções $\rho_1, \rho_2, \ldots, \rho_{d-1}$. Nós usamos as proposições 4.21 e 4.27 para mostrar que com alta probabilidade todas projeções ρ_i são típicas e então argumentamos que se cada ρ_i é típica então $\Psi(f^{m,d})$ é uma fórmula com profundidade 1 e fan-in em torno de qw_d.

Proposição 4.28. *($f^{m,d}$ mantém estrutura)*

Com probabilidade $1 - e^{-\widetilde{\Omega}(w^{1/6})}$ temos que $\Psi(f^{m,d})$ é uma fórmula com profundidade 1 sobre n' variáveis em A_{d-1}, em que $n' \in [qw_d - w^{g(d-1,d)}, qw_d + w^{g(d-1,d)}]$.

Mais especificamente, $\Psi(f^{m,d})$ é o $\vee$ de n' variáveis se d é par ou o $\wedge$ de n' variáves se d é ímpar.

Demonstração. É suficiente argumentar que todas as projeções ρ_i são típicas com alta probabilidade, porque como vimos $\mathrm{Proj}_{\rho_1\rho_2\ldots\rho_i} f^{m,d}$ é uma fórmula com profundidade $d-i$ com fan-in no nível mais baixo no intervalo $[qw - w^{g(i,d)}, qw + w^{g(i,d)}]$. Em particular, $\mathrm{Proj}_{\rho_1\rho_2\ldots\rho_{d-1}} f^{m,d} = \Psi(f^{m,d})$ é uma fórmula com profundidade 1 e fan-in $n' \in [qw_d - w^{g(d-1,d)}, qw_d + w^{g(d-1,d)}]$.

Pelas proposições 4.21 e 4.27 e o Príncipio da Inclusão-Exclusão temos que todas projeções são típicas com probabilidade maior do que $1 - (d-1)e^{-\Omega(w^{1/6})}$, e como d é constante temos que isso é $1 - e^{-\Omega(w^{1/6})}$.

□

Lema da troca para projeções aleatórias

Com o objetivo de mostrar que circuitos de tamanho subexponencial e profundidade $d-1$ simplificam quando aplicamos as projeções $\rho_1, \rho_2, \ldots, \rho_{d-1}$ nós seguimos o mesmo método que usamos quando provamos um limitante inferior para a função paridade e mostramos um lema da troca para projeções aleátorias.

Nós vamos usar a estratégia de prova de Razborov para provar os seguintes lemas:

Lema 4.29. *(Lema da troca para projeções aleatórias $\rho_1 \leftarrow P_1$)*

Seja $F : A_0 \to \{0,1\}$ uma fórmula FNC (ou FND) com largura r, $s \geq 1$ e $\rho_1 \leftarrow P_1$, então

$$\Pr[D_{can}(Proj_{\rho_1} F) \geq s] = \widetilde{\mathcal{O}}(r2^r w^{-1/4})^s.$$

Lema 4.30. *(Lema da troca para projeções aleatórias $\rho_i \leftarrow P_i$)*

Sejam $\rho_1, \rho_2, \ldots, \rho_{i-1}$ tiradas de $P_1, P_2, \ldots, P_{i-1}$ e seja $F : A_{i-1} \to \{0,1\}$ uma fórmula FNC (ou FND) com largura r e tal que $F = Proj_{\rho_1\rho_2\ldots\rho_{i-1}} f$, para alguma função $f : A_0 \to \{0,1\}$ qualquer. Se $s \geq 1$ e $\rho_i \leftarrow P_i(\rho_{i-1})$, então

$$\Pr[D_{can}(Proj_{\rho_1\rho_2\ldots\rho_i} f) \geq s] = \widetilde{\mathcal{O}}(re^{r\frac{t_{i-1}}{1-t_{i-1}}} w^{-1/4})^s.$$

Note que vamos provar um limitante superior para a probabilidade que a *projeção* de F não pode ser expressa por uma árvore de decisão canônica de profundidade pequena que faz consultas às variáveis que representam um bloco da projeção. Como exemplo, para o lema 4.29 nós estamos considerando a probabilidade que $\mathrm{Proj}_{\rho_1} F$ não pode ser computada por uma árvore de decisão de profundidade s que faz consultas às variáveis em A_1.

Nós iremos provar primeiro o lema 4.29 adaptando levemente o processo de codificação da projeção ρ_1, depois provamos o lema 4.30 na única parte em que a prova deste último lema difere da prova do lema anterior. Esta parte corresponde à terceira parte da segunda prova do lema da troca de Håstad em que nós argumentamos que ao estender a projeção ρ_i durante o processo de codificação nós obtemos um aumento no peso da projeção.

Demonstração. (Prova do Lema 4.29)

Nós podemos assumir que $\mathrm{Proj}_{\rho_1} F = \bigwedge_{\alpha=1}^{z} C_\alpha$ é uma fórmula FNC onde cada C_α tem largura no máximo r. Seja $\mathcal{B}_1$ o seguinte subconjunto de projeções tiradas de P_1:

$$\mathcal{B}_1 = \{\rho_1 \leftarrow P_1 | D_{\mathrm{can}}(\mathrm{Proj}_{\rho_1} F) \geq s).$$

Nós vamos novamente codificar $\rho_1 \in \mathcal{B}_1$ mapeando ρ_1 para (ρ_1', μ, β, d) tal que:

- $\rho_1' = \rho_1\sigma$ é uma extensão de ρ_1.
- $\mu \subseteq A_1$ é o subconjunto de blocos "afetados" por σ.
- $\beta \subseteq A_0$ é um subconjunto das variáveis afetadas por σ.
- $\delta \in \{0,1\}^s$ vai ter o mesmo papel que teve na segunda prova do Lema da Troca de Håstad.

Nós dividimos a prova em três partes: na primeira parte iremos descrever o processo de codificação de ρ_1, na segunda parte nós mostramos como recuperar ρ_1 a partir de seu código e na terceira parte nós mostramos um limitante superior para a soma

$$\sum_{\rho_1 \in \mathcal{B}_1} \Pr[\boldsymbol{\rho} = \rho_1], \tag{4.18}$$

obtendo o resultado desejado no enunciado do lema.

1. Construindo o código a partir de ρ_1.

   Nós iremos considerar um caminho π na árvore de decisão canônica de $\mathrm{Proj}_{\rho_1} F$ de tamanho pelo menos s, truncando π de forma que $|\pi| = s$. Seja C_{α_1} a primeira cláusula de $\mathrm{Proj}_{\rho_1} F$ que não foi feita verdadeira por ρ_1, π_1 a porção de π que faz consultas à variáveis que aparecem em C_{α_1}, $\mu_1 \subseteq A_1$ o conjunto de variáveis que aparecem em π_1 e $\beta_1 \subseteq A_0$ o conjunto de variáveis em A_0 que aparecem como um literal negado na cláusula C_{α_1} na fórmula F original, pertencem à alguma variável em μ_1 e foram projetadas pela projeção ρ_1. Nós consideramos a seguinte projeção σ_1 que atribui valores somente às variáveis que pertencem a blocos em μ_1 da seguinte forma. Seja $a \in \mu_1$ e $a' \in \mathrm{input}(a)$:

$$\sigma_1(x_{a'}) = \begin{cases} 1 & \text{se } x_{a'} \in \beta_1. \\ 0 & \text{se } \rho_1(x_{a'}) = x_a \text{ e } x_{a'} \notin \beta_1 \\ x_a & \text{se } \rho_1(x_{a'}) \in \{0,1\}. \end{cases} \tag{4.19}$$

Nós fixamos $\sigma_1(x_a)$ para o valor em $\{0,1\}$ apropriado. Note que definimos σ_1 de forma que C_{α_1} não é satisfeita na fórmula $\text{Proj}_{\rho_1\sigma_1}F$. Nós também temos que $\rho_1\sigma_1$ fixa todas as variáveis que pertecem a algum bloco em μ_1.

Para todos os outros estágios $j = 2, 3, \ldots, l$ nós fazemos a mesma coisa, definindo C_{α_j} como sendo a primeira cláusula não fixada como verdadeira em $\text{Proj}_{\rho\pi_1\pi_2\ldots\pi_{j-1}}F$. De novo nós chegamos no último estágio l quando $\pi_l = \pi \setminus \pi_1\pi_2\ldots\pi_{l-1}$.

Sejam $\sigma = \sigma_1\sigma_2\ldots\sigma_l$, $\rho'_1 = \rho_1\sigma$, $\mu = (\mu_1, \mu_2, \ldots, \mu_l)$ e $\beta = (\beta_1, \beta_2, \ldots, \beta_l)$. Nós definimos a string $\delta \in \{0,1\}^s$ tal que $\delta_j = 1$ se e somente se a j-ésima variável em μ, seguindo a ordem em que estas variáveis aparecem no caminho π, é atribuida um valor diferente por σ e π. Então fazemos $\text{Code}(\rho_1) = (\rho'_1, \mu, \beta, \delta)$.

2. Decodificando $\text{Code}(\rho_1)$.

   A idéia do processo de decodificação de $\text{Code}(\rho_1)$ é basicamente o mesmo que já vimos na segunda prova do Lema da Troca de Håstad. Nós notamos que podemos obter quais variáveis σ_i fixou a partir de β_i, μ_i e σ. Para todo $a \in \mu_i$ e $a' \in \text{input}(a)$ temos que

   $$\sigma_i(x_{a'}) \in \{0,1\} \iff x_{a'} \in \beta_i \text{ ou } \sigma(x_{a'}) = 0.$$

   Que $\sigma_i(x_{a'}) = 1$ se e somente se $x_{a'} \in \beta_i$ segue direto da primeira linha na definição de σ_i em 4.19. Também, como $\rho_1\sigma_1\sigma_2\ldots\sigma_{i-1}$ não atribui o valor 0 para nenhuma variável que pertence a algum bloco em μ_i nós obtemos que $\sigma_i(x_{a'}) = 0$ se e somente se $\sigma(x_{a'}) = 0$. Uma vez que recuperamos σ_i, podemos recuperar π_i com o auxílio da string δ. Daí podemos desfazer σ_i e montar a projeção $\rho_1\pi_1\pi_2\ldots\pi_i\sigma_{i+1}\ldots\sigma_l$ o que nos permite avançar para o $(i+1)$-ésimo estágio. Ao recuperarmos σ podemos extrair ρ_1 de $\rho'_1 = \rho_1\sigma$.

3. Estimando $\Pr[\rho_1 \in \mathcal{B}_1]$.

   Na segunda parte desta prova nós mostramos que o mapeamento Code é injetivo, portanto se mostrarmos que $\Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho} = \rho_1] \leq \kappa \Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho} = \rho'_1]$ para cada $\rho_1 \in \mathcal{B}_1$, em que κ é u um fator $\widetilde{\mathcal{O}}(w^{-s/4})$, nós obtemos um limitante superior para a soma 4.18. Como ρ_1 e ρ'_1 só diferem nos blocos em μ nós iremos nos concentrar primeiro nestes blocos. Seja $a \in \mu$ e ρ_1^a a parte de ρ_1 que atribui valores às variáveis no bloco de a. Como $\rho_1(x_a) = x_a$:

   $$\Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho}^a = \rho_1^a] = q\frac{2^{-m}}{1-2^{-m}}.$$

   Enquanto que

   $$\Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho}^a = \rho_1'^{,a}] = \begin{cases} \lambda & \text{se } \rho'_1(x_a) = 1. \\ (1-\lambda-q)\frac{2^{-m}}{1-2^{-m}} & \text{se } \rho'_1(x_a) = 0. \end{cases}$$

   Lembrando nossa observação que ρ'_1 fixa todas as variáves que pertencem a blocos em μ. Segue de imediato que

   $$\frac{\Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho}^a = \rho_1^a]}{\Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho}^a = \rho_1'^{,a}]} = \begin{cases} \frac{q2^{-m}}{\lambda(1-2^{-m})} & \text{se } \rho'_1(x_a) = 1. \\ \frac{q}{1-\lambda-q} & \text{se } \rho'_1(x_a) = 0. \end{cases}$$

E portanto temos que

$$\Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho}^a = \rho_1^a] \leq \Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho}^a = \rho_1'^{,a}] \times \max\left(\frac{q2^{-m}}{\lambda(1-2^{-m})}, \frac{q}{1-\lambda-q}\right) = \widetilde{\mathcal{O}}(w^{-1/4}) \Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho}^a = \rho_1'^{,a}] \quad (4.20)$$

Como $\Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho} = \rho_1] = \prod_{a\in A_1} \Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho}^a = \rho_1^a]$:

$$\begin{aligned}
\Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho} = \rho_1] &= \prod_{a\in A_1\setminus\mu} \Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho}^a = \rho_1^a] \times \prod_{a\in\mu} \Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho}^a = \rho_1^a] \\
&\leq \prod_{a\in A_1\setminus\mu} \Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho}^a = \rho_1'^{,a}] \times \prod_{a\in\mu} \widetilde{\mathcal{O}}(w^{-1/4}) \Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho}^a = \rho_1'^{,a}] \\
&= \widetilde{\mathcal{O}}(w^{-s/4}) \Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho} = \rho_1'].
\end{aligned}$$

Na primeira desigualdade nós usamos 4.20 e que $\Pr[\boldsymbol{\rho}^a = \rho_1^a] = \Pr[\boldsymbol{\rho}^a = \rho_1'^{,a}]$ sempre que $x_a \in A_1 \setminus \mu$, enquanto que na segunda desigualdade nós usamos que $|\mu| = s$. Agora podemos estimar $\Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho} \in \mathcal{B}_1]$.

$$\begin{aligned}
\Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho} \in \mathcal{B}_1] &= \sum_{\rho_1\in\mathcal{B}} \Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho} = \rho_1] \\
&\leq \sum_{\rho_1\in\mathcal{B}} \widetilde{\mathcal{O}}(w^{-s/4}) \Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho} = \rho_1'].
\end{aligned}$$

Agora nós iremos usar o fato que Code é um mapeamento injetivo. Note primeiro que podemos representar μ, β e δ da seguinte forma:

- Cada μ_i em μ é representada por um vetor em $[r]^{|\mu_i|}$ onde a j-ésima coordenada do vetor guarda a posição da primeira variável pertencendo ao j-ésimo bloco em μ_i na cláusula C_{α_i}. Desta forma representamos μ como uma string de tamanho $s(\log r+1)$ e portanto existem no máximo $2^{s(\log r+1)} = (2r)^s$ possibilidades para μ.
- Cada β_i em β é representado por um vetor em $(\{0,1\}^r)^{|\mu_i|}$ em que cada string de tamanho r neste vetor indica pela posição em que aparece na cláusula C_{α_i} quais variáveis em um dos blocos de μ_i estão em β_i. Podemos então representar β por uma string de tamanho $s(r+1)$ e daí temos que há no máximo $2^{s(r+1)} = (2^{r+1})^s$ possibilidades para β.
- δ é uma string em $\{0,1\}^s$ e portanto existem 2^s possibilidades para δ.

Seja $\mathcal{C}$ o conjunto de todas as restrições ρ tal que $\rho = \rho_1'$ para algum $\rho_1 \in \mathcal{B}_1$. Usando as observações acima a respeito do número de possíveis μ, β e δ e por Code ser um mapeamento injetivo:

$$\begin{aligned}
\sum_{\rho_1\in\mathcal{B}} \widetilde{\mathcal{O}}(w^{-s/4}) \Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho} = \rho_1'] &= \widetilde{\mathcal{O}}(r2^r w^{-1/4})^s \sum_{\rho_1'\in\mathcal{C}} \Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho} = \rho_1'] \\
&= \widetilde{\mathcal{O}}(r2^r w^{-1/4})^s.
\end{aligned}$$

Com isso obtemos $\Pr_{\boldsymbol{\rho}\leftarrow P_1}[\boldsymbol{\rho} \in \mathcal{B}_1] = \widetilde{\mathcal{O}}(r2^r w^{-1/4})^s$ como nós queriamos.

□

Agora iremos provar o Lema 4.30. O mapeamento Code na prova do Lema 4.29 também pode ser usado da mesma forma para mapear uma projeção $\rho_i \leftarrow P_i$ para (ρ_i', μ, β, d) com a diferença que estamos consideranto variáves de entradas em A_i para a função $\text{Proj}_{\rho_1\rho_2\ldots\rho_i}F$. O que realmente muda é que temos que adaptar alguns cálculos levando em conta a forma que P_i distribui pesos para as projeções $\rho_i \leftarrow P_i$.

Demonstração. (Prova do Lema 4.30.)

Nós vamos considerar que $F = \text{Proj}_{\rho_1\rho_2\ldots\rho_{i-1}}f = \bigwedge_{\alpha=1}^{z} C_\alpha$ é uma fórmula FNC em que cada cláusula contém no máximo r variáveis. Desta vez nós temos o conjunto $\mathcal{B}_i$ definido como:

$$\mathcal{B}_i = \{\rho_i \leftarrow P_i(\rho_{i-1}) | D_{\text{can}}(\text{Proj}_{\rho_i}F) \geq s\},$$

em que $\text{Proj}_{\rho_i}F = \text{Proj}_{\rho_1\rho_2\ldots\rho_i}f$. Também iremos assumir que i é ímpar (a prova para o caso em que i é par é perfeitamente análoga com os papeis de 0 e 1 trocados). Seja $\rho_i \in \mathcal{B}_i$ e para cada $a \in \mu$ seja S_a como definido em 4.22. Note que por ρ' ter alterado o bloco a nós necessariamente devemos ter $\big||S_a| - qw\big| \leq w^{g(i-1,d)}$, pois senão a projeção ρ_i tirada de $P_i(\rho_{i-1})$ teria fixada todas as variáveis em input(a) com um valor constante. Seja $s_1(a)$ o número de variáveis em S_a que foram atribuidas o valor 1 por ρ_i, $s_1'(a)$ o número de variáveis em S_a que foram fixadas com o valor 1 na extensão ρ_i' de ρ_i e $\Delta_a = s_1'(a) - s_1(a) \geq 0$. Definindo ρ_i^a como sendo a parte de ρ_i que atribui valores às variáveis no bloco de a, nós temos que

$$\Pr_{\boldsymbol{\rho}\leftarrow P_i}[\boldsymbol{\rho}^a = \rho_i^a] = q_a \frac{t_{i-1}^{|S_a|-s^1(a)}(1-t_{i-1})^{s_1(a)}}{1-(1-t_{i-1})^{|S_a|}},$$

onde q_a é o mesmo definido em 4.22. Também,

$$\Pr_{\boldsymbol{\rho}\leftarrow P_i}[\boldsymbol{\rho}^a = \rho_i'^{,a}] = \begin{cases} \lambda & \text{se } \rho_i'(x_a) = 1. \\ (1-\lambda-q_a)\frac{t_{i-1}^{|S_a|-s_1'(a)}(1-t_{i-1})^{s_1'(a)}}{1-(1-t_{i-1})^{|S_a|}} & \text{se } \rho_i'(x_a) = 0. \end{cases}$$

E portanto,

$$\frac{\Pr_{\boldsymbol{\rho}\leftarrow P_i}[\boldsymbol{\rho}^a = \rho_i^a]}{\Pr_{\boldsymbol{\rho}\leftarrow P_i}[\boldsymbol{\rho}^a = \rho_i'^{,a}]} = \begin{cases} \frac{q_a}{\lambda}\frac{t_{i-1}^{\Delta_a}(1-t_{i-1})^{|S_a|-\Delta_a}}{1-(1-t_{i-1})^{|S_a|}} & \text{se } \rho_i'(x_a) = 1 \\ \frac{q_a}{1-\lambda-q_a}\left(\frac{t_{i-1}}{1-t_{i-1}}\right)^{\Delta_a} & \text{se } \rho_1'(x_a) = 0, \end{cases}$$

onde nós usamos que $s_1'(a) = |S_a|$ sempre que $\rho_i'(x_a) = 1$. Daí obtemos que

$$\Pr_{\boldsymbol{\rho}\leftarrow P_i}[\boldsymbol{\rho}^a = \rho_i^a] \leq \Pr_{\boldsymbol{\rho}\leftarrow P_i}[\boldsymbol{\rho}^a = \rho_i'^{,a}] \times \max\left(\frac{q_a}{\lambda}\frac{t_{i-1}^{\Delta_a}(1-t_{i-1})^{|S_a|-\Delta_a}}{1-(1-t_{i-1})^{|S_a|}}, \frac{q_a}{1-\lambda-q_a}\left(\frac{t_{i-1}}{1-t_{i-1}}\right)^{\Delta_a}\right).$$

Nós queremos um limitante superior para ambas expressões que aparecem dentro do max. Como temos que $qw - w^{g(i-1,d)} \leq |S_a| \leq qw + w^{g(i-1,d)}$:

$$(1-t_{i-1})^{qw+w^{g(i-1,d)}} \leq (1-t_{i-1})^{|S_a|} \leq (1-t_{i-1})^{qw-w^{g(i-1,d)}},$$

o que implica em $(1-t_{i-1})^{|S_a|} = 2^{-m}(1 \pm o(1))$. Daí, relembrando a relação entre q_a e q da proposição 4.23 e que $\frac{q2^{-m}}{\lambda} = \widetilde{\mathcal{O}}(w^{-1/4})$, temos que

$$\begin{aligned}\frac{q_a}{\lambda}\frac{t_{i-1}^{\Delta_a}(1-t_{i-1})^{|S_a|-\Delta_a}}{1-(1-t_{i-1})^{|S_a|}} &= \frac{q_a}{\lambda}\frac{(1-t_{i-1})^{|S_a|}}{1-(1-t_{i-1})^{|S_a|}}\left(\frac{t_{i-1}}{1-t_{i-1}}\right)^{\Delta_a} \\ &= \frac{q_a}{\lambda}2^{-m}(1\pm o(1))\left(\frac{t_{i-1}}{1-t_{i-1}}\right)^{\Delta_a} \\ &= \widetilde{\mathcal{O}}(w^{-1/4})\left(\frac{t_{i-1}}{1-t_{i-1}}\right)^{\Delta_a}.\end{aligned}$$

Também temos que $\frac{q_a}{1-\lambda-q_a} = \widetilde{\mathcal{O}}(w^{-1/2}) = \widetilde{\mathcal{O}}(w^{-1/4})$ e portanto podemos concluir que

$$\Pr_{\boldsymbol{\rho}\leftarrow P_i}[\boldsymbol{\rho}^a = \rho_i^a] = \widetilde{\mathcal{O}}(w^{-1/4})\left(\frac{t_{i-1}}{1-t_{i-1}}\right)^{\Delta_a}\Pr_{\boldsymbol{\rho}\leftarrow P_i}[\boldsymbol{\rho}^a = \rho_i'^{,a}].$$

Seja $\|\beta\| = \sum_{a\in\mu}\Delta_a$ o número total de variáveis que foram fixadas em 1 por σ, ou equivalentemente o peso de Hamming de β. Então temos que

$$\Pr_{\boldsymbol{\rho}\leftarrow P_i}[\boldsymbol{\rho} = \rho_i] = \widetilde{\mathcal{O}}(w^{-s/4})\left(\frac{t_{i-1}}{1-t_{i-1}}\right)^{\|\beta\|}\Pr_{\boldsymbol{\rho}\leftarrow P_i}[\boldsymbol{\rho} = \rho_i'], \tag{4.21}$$

para todo $\rho_i \in \mathcal{B}_i$. Definindo o conjunto $\mathcal{C}$ da mesma forma que fizemos na prova do Lema 4.29 e somando sobre todas as possibilidades de β obtemos o seguinte:

$$\sum_{\beta}\sum_{\rho_i'\in\mathcal{C}}\left(\frac{t_{i-1}}{1-t_{i-1}}\right)^{\|\beta\|}\Pr_{\boldsymbol{\rho}\leftarrow P_i}[\boldsymbol{\rho} = \rho_i'] \le \sum_{k=0}^{rs}\binom{rs}{k}\left(\frac{t_{i-1}}{1-t_{i-1}}\right)^{k}\Pr_{\boldsymbol{\rho}\leftarrow P_i}[\boldsymbol{\rho}\in\mathcal{C}] \le \left(1+\frac{t_{i-1}}{1-t_{i-1}}\right)^{rs} \le e^{rs\frac{t_{i-1}}{1-t_{i-1}}}. \tag{4.22}$$

Se somarmos também sobre todas as possibilidades de μ e δ e levando em conta as desigualdades 4.21 e 4.22 temos que:

$$\Pr_{\boldsymbol{\rho}\leftarrow P_i}[\boldsymbol{\rho}\in\mathcal{B}_i] = \widetilde{\mathcal{O}}(re^{r\frac{t_{i-1}}{1-t_{i-1}}}w^{-1/4})^s,$$

o que conclui a prova do lema.

□

Prova dos Teoremas 3.12 e 3.14

Agora nós iremos provar o Teorema 3.12 que nós enunciamos mais uma vez por conveniência.

Teorema 4.31. *Seja $d > 2$ e m suficientemente grande, então qualquer circuito de tamanho $S \le 2^{w^{1/5}}$ e profundidade $d-1$ não computa a função $f^{m,d}$ corretamente em todas as entradas.*

Teorema 4.32. *Seja $d > 2$ e m suficientemente grande, então qualquer circuito de tamanho $S \le 2^{w^{1/5}}$ e profundidade $d-1$ falha em computar a função $f^{m,d}$ corretamente numa fração maior do que $1/2 - \widetilde{\mathcal{O}}(w^{-1/2})$ das entradas.*

O Teorema 4.31 é consequência de dois fatos que listamos a seguir. Seja C um circuito de profundidade d com no máximo $2^{w^{1/5}}$ portas lógicas no seu segundo nível pra cima e com fan-in $m/5$ no seu nível mais baixo:

1. Com alta probabilidade, $\Psi(C) = \mathrm{Proj}_{\rho_1\rho_2\dots\rho_{d-1}} C$ é computada por uma árvore de decisão de profundidade pequena.
2. Com alta probabilidade, $\Psi(f^{m,d}) = \mathrm{Proj}_{\rho_1\rho_2\dots\rho_{d-1}} f^{m,d}$ é uma fórmula de profundidade 1 sobre um número razoável de variáveis em A_{d-1}.

O item (2) é a Proposição 4.28. O item (1) nós usamos a mesma estratégia na prova do Teorema 4.12 usando uma sequência de aplicações dos Lemas 4.29 e 4.30 para reduzir a profundidade do circuito. Note que estamos agora falando de circuitos com profundidade d, mas a proposição 4.31 segue pois qualquer circuito de profundidade $d-1$ pode trivialmente ser convertido em um circuito de profundidade d com fan-in arbitrário se trocarmos cada variável de entrada que alimenta uma porta $\wedge$ ($\vee$) no seu nível mais baixo por uma porta $\vee$ ($\wedge$) que recebe como entrada somente aquela variável de entrada.

Proposição 4.33. *Seja $d > 2$, então qualquer circuito C de profundidade d com no máximo $2^{w^{1/5}}$ portas no seu segundo nível pra cima e fan-in $m/5$ no nível mais baixo satisfaz:*

$$\Pr[D(\Psi(C)) \leq 10] \geq 1 - \widetilde{\mathcal{O}}(w^{-1/2}),$$

onde $\Psi = Proj_{\rho_1\rho_2\dots\rho_{d-1}}$ em que cada $\rho_i \leftarrow P_i$.

Demonstração. Vamos denotar por S_i o número de portas lógicas no i-ésimo nível de C e $S^* = \sum_{i=2}^{d} S_i \leq 2^{w^{1/5}}$.

Vamos assumir sem perda de generalidade que as portas lógicas no nível mais baixo de C são portas $\wedge$. O que nós queremos fazer é aplicar os lemas 4.29 e 4.30 em cada nível do circuito, assim como fizemos na prova do teorema 3.8, reduzindo a profundidade do circuito em 1 com alta probabilidade. Agora mostramos como proceder com este argumento.

- Primeiro nós aplicamos o lema da troca para o espaço de projeções P_1 (4.29) com os parâmetros $r = m/5$ e $s = \log S^{*2}$. Seja a uma porta $\vee$ no segundo nível de C e seja f_a a fórmula FND computada por a. Pelo lema da troca para P_1 e por C ter fan-in $m/5$ no seu nível mais baixo, nós temos o seguinte.

  $$\Pr[D_{\mathrm{can}}(\mathrm{Proj}_{\rho_1} f_a) \geq \log S^{*2}] = \widetilde{\mathcal{O}}\Big(\frac{m}{5} 2^{m/5} w^{-1/4}\Big)^{\log S^{*2}}.$$

  E como $\frac{m}{5} 2^{m/5} w^{-1/4} = \widetilde{\mathcal{O}}(w^{-1/20}) = o(1)$ temos que para m suficientemente grande (lembrando que w depende de m) é verdade que

  $$\Pr[D_{\mathrm{can}}(\mathrm{Proj}_{\rho_1} f_a) \geq \log S^{*2}] \leq 2^{-\log S^{*2}} = \frac{1}{S^{*2}}.$$

- Agora, seja $i \in \{2, 3, \dots, d-2\}$ e assuma que as projeções $\rho_1, \rho_2, \dots, \rho_{i-1}$ sucederam em todas as suas aplicações. Nós aplicamos o lema da troca para o espaço de projeção P_i com os parâmetros $r = s = \log S^{*2}$. Seja a uma porta lógica no $(i+1)$-ésimo nível de C que computa f_a que no caso seria uma fórmula FND se i é ímpar ou uma fórmula FNC se i é par. Por 4.30 e pelas portas lógicas no i-ésimo nível agora terem fan-in no máximo $\log S^{*2}$ nós temos que

  $$\Pr[D_{\mathrm{can}}(\mathrm{Proj}_{\rho_1\rho_2\dots\rho_i} f_a) \geq \log S^{*2}] = \widetilde{\mathcal{O}}\big(\log S^{*2} e^{\log S^{*2} \frac{t_{i-1}}{1-t_{i-1}}} w^{-1/4}\big)^{\log S^{*2}}.$$

Nós temos de novo que a expressão na base da exponenciação é $\widetilde{\mathcal{O}}(w^{-1/20}) = o(1)$ e portanto para m suficientemente grande,

$$\Pr[D_{\text{can}}(\text{Proj}_{\rho_1\rho_2\ldots\rho_i} f_a) \geq \log S^{*2}] \leq 2^{-\log S^{*2}} = \frac{1}{S^{*2}}.$$

- Para o último passo assuma que todas projeções anteriores sucederam e portanto o que resta é uma fórmula de profundidade 2 sobre as variáveis em A_{d-2}. Nós aplicamos o lema de troca para o espaço de projeções P_{d-1} às variáves de A_{d-2} com os parâmetros $r = \log S^{*2}$ e $r = 10$ e temos que

$$\Pr[D_{\text{can}}(\Psi(C)) \geq 10] = \widetilde{\mathcal{O}}\big(\log S^{*2} e^{\log S^{*2} \frac{t_{i-1}}{1-t_{i-1}}} w^{-1/4}\big)^{10}, \tag{4.23}$$

o que implica em

$$\Pr[D_{\text{can}}(\Psi(C)) \geq 10] = \widetilde{\mathcal{O}}(w^{-1/2}),$$

porque a expressão na base da exponenciação em 4.23 é $\widetilde{\mathcal{O}}(w^{-1/20})$.

Por fim aplicamos o princípio da inclusão-exclusão para dar um limitante inferior para a probabilidade que todos as projeções sucederam em transformar as fórmulas de profundidade 2 computada por cada porta lógica de C em uma árvore de decisão de baixa profundidade, obtendo o seguinte.

$$\Pr[D_{\text{can}}(\Psi(C)) < 10] \geq 1 - \frac{S^*}{S^{*2}} - \widetilde{\mathcal{O}}(w^{-1/2}) \geq 1 - \frac{1}{2^{w^{1/5}}} - \widetilde{\mathcal{O}}(w^{-1/2}) = 1 - \widetilde{\mathcal{O}}(w^{-1/2}).$$

□

Então nós já temos por 4.28 e 4.33 que $f^{m,d}$ não pode ser computada por um circuito de tamanho $2^{w^{1/5}}$, profundidade d e fan-in $m/5$ no nível mais baixo pois com probabilidade positiva teremos que $\Psi(f^{m,d})$ é uma função que depende em um grande número de variáveis enquanto que $\Psi(C)$ depende apenas de um número constante de variáveis. Por completude a prova do teorema 4.31 segue abaixo:

Demonstração. (Prova do teorema 3.12, 4.31)

Seja C um circuito de profundidade d, tamanho menor do que $2^{w^{1/5}}$ e com fan-in no nível mais baixo menor do que $m/5$. Por 4.33 sabemos que com probabilidade maior do que $1 - \widetilde{\mathcal{O}}(w^{-1/2})$, $\Psi(C)$ é representável por uma árvore de decisão de profundidade no máximo 10. Por outro lado, por 4.28 temos que com probabilidade maior do que $1 - e^{-\widetilde{\Omega}(w^{1/6})}$ nós temos que $\Psi(f^{m,d})$ é um circuito de profundidade 1 que depende de n' variáveis, onde $n' \in [qw_d - w^{g(d-1,d)}, qw_d + w^{g(d-1,d)}]$. Portanto, pelo Princípio da Inclusão-Exclusão com probabilidade maior do que $1 - \widetilde{\mathcal{O}}(w^{-1/2}) - e^{-\widetilde{\Omega}(w^{1/6})}$ temos que ambos os eventos acontecem. Como o $\vee$ de $n' >> 10$ variáveis não pode ser representado por um árvore de decisão de profundidade 10, temos pelo método probabilístico que C não pode ser um circuito para a função $f^{m,d}$.

□

Para provar o teorema 4.32 nosso objetivo agora é mostrar que $\Psi(f^{m,d})$ e $\Psi(C)$ com alta probabilidade nem sequer são correlacionadas.

Proposição 4.34. *Seja $f : A_{d-1} \to \{0,1\}$ o $\vee$ de n' variáveis, onde $n' \in [qw_d - w^{g(d-1,d)}, qw_d + w^{g(d-1,d)}]$, e $g : A_{d-1} \to \{0,1\}$ uma função computável por uma árvore de decisão de profundidade 10, então temos que*

$$\Pr_{\boldsymbol{x} \sim \{0_{1-t_{d-1}}, 1_{t_{d-1}}\}}[f(\boldsymbol{x}) \neq g(\boldsymbol{x})] \geq 1/2 - \widetilde{\mathcal{O}}(w^{-1/12}).$$

Demonstração. Seja T uma árvore de decisão com profundidade 10 que melhor aproxima f. Para alguma entrada $x \leftarrow \{0_{1-t_{d-1}}, 1_{t_{d-1}}\}^{A_{d-1}}$ temos que T vê um 1 com probabilidade no máximo $10t_{d-1}$ e nesse caso T pode sempre dar como saída o valor correto de $f(x)$ (no caso teriamos $T(x) = f(x) = 1$).

Por outro lado, com probabilidade maior do que $1 - 10t_{d-1}$, T vê somente 0s após fazer 10 consultas às suas variáveis de entrada. Vamos considerar então a função f' que é o $\vee$ das $n' - 10$ variáveis em A_{d-1} que não foram consultadas por T no caso em que todas as 10 consultas retornaram 0. Então, a melhor estratégia que T pode fazer é adivinhar o valor de f' sobre uma entrada tirada da distribuição $\{0_{1-t_{d-1}}, 1_{t_{d-1}}\}^{A_{d-1}}$. Mais especificamente, a melhor estratégia que T pode tomar é dar como saída um valor $b \in \{0, 1\}$ tal que com respeito à distribuição $\{0_{1-t_{d-1}} 1_{t_{d-1}}\}^{A_{d-1}}$ é verdade que $\Pr[f' = b] \geq \Pr[f' = 1 - b]$. Nós temos que

$$\Pr_{\boldsymbol{x} \sim \{0_{1-t_{d-1}}, 1_{t_{d-1}}\}}[f'(\boldsymbol{x}) = 0] = (1 - t_{d-1})^{n'-10},$$

o qual podemos ver satisfaz

$$1/2 - \widetilde{\mathcal{O}}(w^{-1/12}) \leq (1 - t_{d-1})^{n'-10} \leq 1/2 + \widetilde{\mathcal{O}}(w^{-1/12}).$$

Segue então que qualquer que for a estratégia que T pode tomar, T irá dar como saída o valor correto de $f(x)$ com probabilidade no máximo $1/2 + \widetilde{\mathcal{O}}(w^{-1/12})$. Ao todo, T corretamente computa f com probabilidade no máximo

$$10t_{d-1} + 1/2 + \widetilde{\mathcal{O}}(w^{-1/12}) = 1/2 + \widetilde{\mathcal{O}}(w^{-1/12}).$$

Podemos então concluir que

$$\Pr_{\boldsymbol{x} \sim \{0_{1-t_{d-1}}, 1_{t_{d-1}}\}}[f(\boldsymbol{x}) \neq T(\boldsymbol{x})] \geq 1/2 - \widetilde{\mathcal{O}}(w^{-1/12}).$$

Pela forma que definimos T o resultado da proposição segue como consequência.

□

A partir de 4.33, 4.28 e 4.34 podemos provar o teorema 4.32.

Demonstração. (Prova do teorema 3.14, 4.32)

Na prova do teorema 4.31 nós vimos que com probabilidade $1 - \widetilde{\mathcal{O}}(w^{-1/2})$ é verdade que $\Psi(C)$ e $\Psi(f^{m,d})$ caem no caso da proposição 4.34. Portanto temos que

$$E_\Psi\left[\Pr_{\boldsymbol{x} \sim \{0_{1-t_{d-1}}, 1_{t_{d-1}}\}}[\Psi(C) \neq \Psi(f^{m,d})]\right] \geq \left(1 - \widetilde{\mathcal{O}}(w^{-1/2})\right)\left(1/2 - \widetilde{\mathcal{O}}(w^{-1/12})\right) = 1/2 - \widetilde{\mathcal{O}}(w^{-1/2}).$$

Daí segue por 4.25 que

$$\Pr_{\boldsymbol{x} \sim \{0,1\}^n}[C(\boldsymbol{x}) \neq f^{m,d}(\boldsymbol{x})] \geq 1/2 - \widetilde{\mathcal{O}}(w^{-1/2}).$$

□

Capítulo 5

Teoremas de transferência

Logo no começo da introdução deste trabalho nós mencionamos uma dicotomia entre complexidade computacional, o estudo de quão difíceis problemas computacionais são, e o design de algoritmo eficientes. Pode parecer que achar um algoritmo eficiente para um problema não nos diria nada a respeito da dificuldade de outros problemas, porém, a existência de hierarquias de classes de complexidade como as que vimos em (...) na verdade impõem limitações para a existência de algoritmos rápidos para determinados problemas e colapsos de algumas classes de complexidade. Para nos convecermos disto nós podemos considerar o seguinte teorema:

Teorema 5.1 (Meyer)**.** *Se* $\mathsf{EXP} \subseteq \mathsf{P/poly}$ *então* $\mathsf{EXP} = \Sigma_2^p$.

Agora, assuma que $EXP \subseteq \mathsf{P/poly}$ e $\mathsf{P} = \mathsf{NP}$. Nós podemos generalizar o teorema 2.29 e obter que $\mathsf{P} = \mathsf{NP}$ implica em $\mathsf{P} = \mathsf{PH} = \Sigma_2^p$, e então pelo teorema 5.1 nós teriamos que $\mathsf{P} = \mathsf{EXP}$, o que é uma contradição pelo teorema da hierarquia de tempo determinístico que nós vimos em 2.43. Portanto, achar um circuito de tamanho polinomial para um problema EXP-completo nós daria um limitante inferior para o tempo necessário para decidir um problema NP-completo como SAT. O que está por trás deste exemplo e de todos outros resultados deste tipo que veremos é que uma vez que não sabemos a relação exata entre P, Σ_2^p e EXP, com a exceção que sabemos que EXP não pode estar contido em P, então dizer que EXP é tão "fácil" quanto Σ_2^p também nos diz sobre a perspectiva da classe P que Σ_2^p não pode ser uma classe tão fraca quanto P, e daí obtemos que $\mathsf{P} \neq NP$.

Antes de falar sobre teoremas de transferência nós iremos ver o porquê precisamos destas técnica, ou equivalentemente iremos ver o porquê que o método de restrições/projeções aleatórias do capítulo anterior não são capazes de provar limitantes inferiores para classes maiores do que AC^0.

5.1 Provas Naturais

Referências Bibliográficas

[AB09] Sanjeev Arora and Boaz Barak, *Computational complexity: a modern approach*, Cambridge University Press, 2009.

[Bea94] Paul Beame, *A switching lemma primer*, Tech. report, Technical Report UW-CSE-95-07-01, Department of Computer Science and Engineering, University of Washington, 1994.

[Coo71] Stephen A Cook, *The complexity of theorem-proving procedures*, Proceedings of the third annual ACM symposium on Theory of computing, ACM, 1971, pp. 151–158.

[Coo73] ———, *A hierarchy for nondeterministic time complexity*, Journal of Computer and System Sciences **7** (1973), no. 4, 343–353.

[Edm65] Jack Edmonds, *Paths, trees, and flowers*, Canadian Journal of mathematics **17** (1965), no. 3, 449–467.

[FM05] Gudmund Skovbjerg Frandsen and Peter Bro Miltersen, *Reviewing bounds on the circuit size of the hardest functions*, Information Processing Letters **95** (2005), no. 2, 354–357.

[For94] Lance Fortnow, *The role of relativization in complexity theory*, Bulletin of the EATCS **52** (1994), 229–243.

[For09] ———, *The status of the p versus np problem*, Communications of the ACM **52** (2009), no. 9, 78–86.

[GJ02] Michael R Garey and David S Johnson, *Computers and intractability*, vol. 29, wh freeman, 2002.

[Gol00] Oded Goldreich, *Computational complexity*, In The Princeton Companion, Citeseer, 2000.

[Hås87] Johan Håstad, *Computational limitations of small-depth circuits*.

[HS65] Juris Hartmanis and Richard E Stearns, *On the computational complexity of algorithms*, Transactions of the American Mathematical Society (1965), 285–306.

[HS66] Fred C Hennie and Richard Edwin Stearns, *Two-tape simulation of multitape turing machines*, Journal of the ACM (JACM) **13** (1966), no. 4, 533–546.

[Kar72] Richard M Karp, *Reducibility among combinatorial problems*, Springer, 1972.

[KL82] Richard M Karp and Richard Lipton, *Turing machines that take advice*, Enseign. Math **28** (1982), no. 2, 191–209.

[Ko] Ker-I Ko, *Constructing oracles by lower bound techniques for circuits*.

[Lev73] Leonid A Levin, *Universal sequential search problems*, Problemy Peredachi Informatsii **9** (1973), no. 3, 115–116.

[LP97] Harry R Lewis and Christos H Papadimitriou, *Elements of the theory of computation*, Prentice Hall PTR, 1997.

[Lup58] Oleg B Lupanov, *A method of circuit synthesis*, Izvestia vuz Radio zike **1** (1958), 120–140.

[RR94] Alexander A Razborov and Steven Rudich, *Natural proofs*, Proceedings of the twenty-sixth annual ACM symposium on Theory of computing, ACM, 1994, pp. 204–213.

[RST15a] Benjamin Rossman, Rocco A Servedio, and Li-Yang Tan, *An average-case depth hierarchy theorem for boolean circuits*, Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on, IEEE, 2015, pp. 1030–1048.

[RST15b] Benjamin Rossman, Rocco A. Servedio, and Li-Yang Tan, *Complexity theory column 89: The polynomial hierarchy, random oracles, and boolean circuits*, SIGACT News **46** (2015), no. 4, 50–68.

[S+49] Claude Shannon et al., *The synthesis of two-terminal switching circuits*, Bell System Technical Journal **28** (1949), no. 1, 59–98.

[Sav70] Walter J Savitch, *Relationships between nondeterministic and deterministic tape complexities*, Journal of computer and system sciences **4** (1970), no. 2, 177–192.

[Sav98] John E Savage, *Models of computation*, Exploring the Power of Computing (1998).

[Sip12] Michael Sipser, *Introduction to the theory of computation*, Cengage Learning, 2012.

[Sub61] Bella Abramovna Subbotovskaya, *Realizations of linear functions by formulas using+*, Doklady Akademii Nauk SSSR **136** (1961), no. 3, 553–555.

[Tur36] Alan Mathison Turing, *On computable numbers, with an application to the entscheidungsproblem*, J. of Math **58** (1936), 345–363.

[Wil13] Ryan Williams, *Improving exhaustive search implies superpolynomial lower bounds*, SIAM Journal on Computing **42** (2013), no. 3, 1218–1244.

[Wil14] ———, *Nonuniform acc circuit lower bounds*, Journal of the ACM (JACM) **61** (2014), no. 1, 2.