18 de outubro de 2016

Fórmulas FNC e FND

Uma fórmula FNC F sobre as variáveis $x_1, \ldots, x_n$ é a conjunção de disjunções de literais (uma das variáveis ou a tua negação).

$$F = C_1 \wedge \cdots \wedge C_s$$

Onde cada cláusula C_i é a disjunção de literais e s é o tamanho de F.
Se cada clásula tem no máximo k literais então dizemos que F tem largura k e dizemos que F é uma k-FNC.

Fórmulas FNC e FND

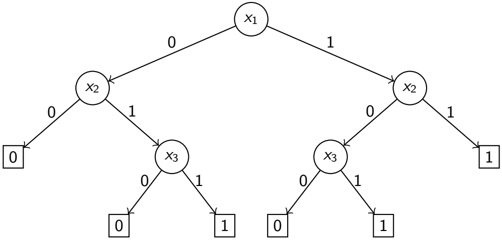
Uma fórmula FND F por outro lado é a disjunção de conjuções de literais.

$$F = T_1 \vee \cdots \vee T_s$$

Onde cada termo T_i é a conjunção de literais e s é o tamanho de F.
De novo, se cada termo tem no máximo k literais então F tem largura k e F é uma k-FND.

Árvores de decisão

Uma árvore de decisão é algo como a imagem abaixo:



- Cada nodo leva o label de uma das variáveis.
- Começando do nodo mais alto, o algoritmo ramifica para a direita ou à esquerda dependendo do valor da variável lida.
- As folhas guardam o valor da função em cada entrada que chega nela.

Árvores de decisão

Denotamos a sáida de uma árvore de decisão T sobre a entrada x por $T(x)$. Se f é tal que $f(x) = T(x)$ para todos os x então dizemos que T computa a função f.

- Por exemplo, a árvore de decisão do slide anterior computa a função que Majority_3, que é 1 se e somente se o número de 1s na entrada é pelo menos 2.

O tamanho de T é o número de folhas e tua profundidade é o maior camnho do nodo mais alto até uma das folhas.

- A árvore do slide anterior tem tamanho 6 e profundidade 3.

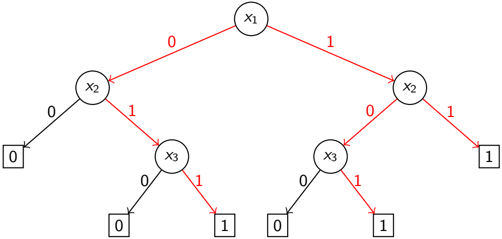
Árvores de decisão

É importante notar que árvores de decisão são mais fracas do que fórmulas FNC (FND).

- Se T é uma árvore de decisão de tamanho s e profundidade d então existe uma fórmula FND (FNC) F tal que $F(x) = T(x)$, para todos x, de tamanho $\leq s$ e largura $\leq d$.
  - FND: Cada caminho P da árvore tal que $T(P) = 1$ define uma cláusula.
  - FNC: Cada caminho P da árvore tal que $T(P) = 0$ define um termo.

Árvores de decisão

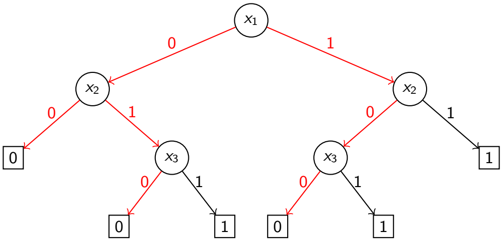
É importante notar que árvores de decisão são mais fracas do que fórmulas FNC (FND).



$$F = (\overline{x}_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \overline{x}_2 \wedge x_3) \vee (x_1 \wedge x_2)$$

Árvores de decisão

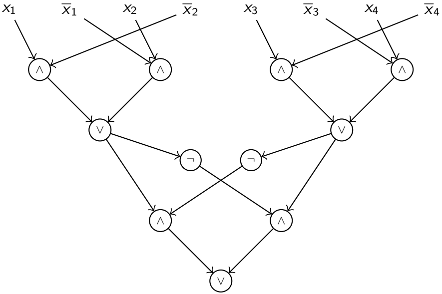
É importante notar que árvores de decisão são mais fracas do que fórmulas FNC (FND).



$$F = (x_1 \vee x_2) \wedge (x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3)$$

Circuitos

Um circuito Booleano é composto de portas lógicas computando uma das funções em $\{\wedge, \vee, \neg)$ e fios ligando estas portas lógicas como mosta a figura abaixo:



Este circuitos tem 4 variáveis de entrada (sem contar a negação de cada variável). Todos circuitos tem uma única porta lógica no nível mais alto, o valor desta porta lógica é a saída do circuito.

Circuitos

Nós dizemos que um circuito C com n variáveis de entrada computa $f : \{0,1\}^n \to \{0,1\}$ se $C(x) = f(x)$, para todos $x \in \{0,1\}^n$.

- O circuito do slide anterior computa a função Parity_4.
  - $\text{Parity}_4(x) = 1$ se e somente se $x_1 + x_2 + x_3 + x_4 \equiv 1 \pmod 2$.

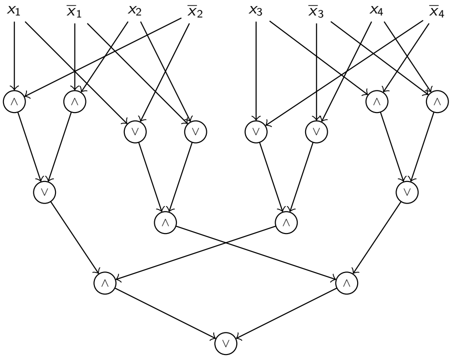
O tamanho de um circuito é o número de portas lógicas e a tua profundidade é o tamanho do maior caminho de uma variável de entrada até a porda de saída.

- O nosso circuito para $Parity_4$ tem tamanho 11 e profundidade 4.
- Em geral, Parity_n tem um circuito de tamanho $\mathcal{O}(n)$ e profundidade $\mathcal{O}(\log n)$.

Circuitos

Pela lei de De Morgan nós podemos empurrar as portas $\neg$ para as variáveis de entrada.

- Se o circuito original tinha tamanho S então o circuito resultante tem tamanho $\leq 2S$.



O circuito acima também computa Parity_4 e tem tamanho 15.

Complexidade de circuitos

Dada uma função $f : \{0,1\}^n \to \{0,1\}$ nós queremos saber qual é o menor circuito que computa f.

- Seja $\mathscr{C}$ o conjunto de circuitos que computam f.
- $\text{Size}(f) = \min_{C \in \mathscr{C}}\{|C|\}$.

Se $f : \{0,1\}^* \to \{0,1\}$ então temos que definir uma sequência de circuitos $\{C_n\}_{n\geq 1}$ onde cada C_n computa f restrita à strings de tamanho n.

- $\text{Size}(f) = \mathcal{O}(g)$ se existem constantes c e n_0 tal que $|C_n| \leq cg(n)$, para todos $n \geq n_0$.
- Como já comentamos, $\text{Size}(\text{Parity}) = \mathcal{O}(n)$.

Complexidade de circuitos: P/poly

Algumas classes de complexidade de circuitos:

- P/poly : circuitos de tamanho polinomial.
  - Contém toda a classe P.
  - Contém todas as linguagens unárias.
    - Logo contém alguns problemas indecidíveis.
  - A tua versão (P-)uniforme (ou logspace-uniforme) coincide com a classe P.

Complexidade de circuitos: P/poly

- Problema em aberto: $\mathrm{NP} \subseteq \mathrm{P/poly}$?
  - $\mathrm{NP} \not\subseteq \mathrm{P/poly}$ implicaria em $\mathrm{P} \neq \mathrm{NP}$.
  - Teorema de Karp-Lipton: $\mathrm{NP} \subseteq \mathrm{P/poly} \Rightarrow \mathrm{PH} = \Sigma_2^p$.
- Problema em aberto: Existe, para todo $k \geq 1$, uma linguagem em P que não admite circuitos de tamanho n^k?
  - Suponha que $\mathrm{P} \neq \mathrm{NP}$, isto é verdade porque a classe NP não admite circuitos pequenos ou é porque circuitos para problemas em P são pequenos demais?

Complexidade de circuitos: NC e AC

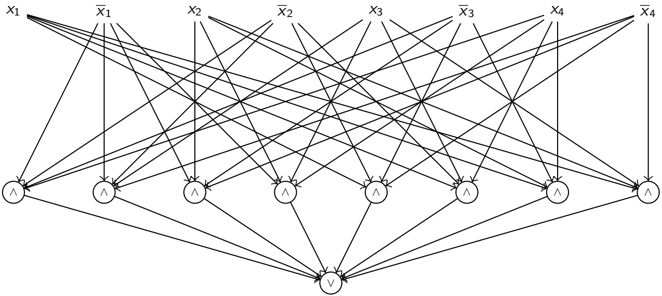
Algumas classes de complexidade de circuitos:

- NC^i: circuitos de tamanho polinomial e profundidade $\log^i n$.
- $\mathsf{NC} = \bigcup_{i \geq 0} \mathsf{NC}^i$.
  - Exemplo: computar a determinante de uma matriz está em NC.
- AC^i: circuitos de tamanho polinomial, profundidade $\log^i n$ e fan-in arbitrário.
- $\mathsf{AC} = \bigcup_{i \geq 0} \mathsf{AC}^i$.
- $\mathsf{NC}^0 \subseteq \mathsf{AC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{AC}^1 \subseteq \cdots \Rightarrow \mathsf{NC} = \mathsf{AC}$.

Complexidade de circuitos: NC e AC

Um circuito AC^0 é qualquer circuito com fn-in arbitrário e que alterna portas $\vee$ e portas $\wedge$.

- Por exemplo, o seguite circuito é um circuito AC^0 para Parity_4.



- Todas fórmulas FNC e FND são circuitos AC^0.

Tribes

A função $\text{Tribes}_{w,s} : \{0,1\}^{ws} \to \{0,1\}$ é definida da seguinte forma:

$$\text{Tribes}_{w,s}(x) = \bigvee_{i=1}^{s} (x_{1,i} \wedge x_{2,i} \wedge \cdots \wedge x_{w,i}).$$

Onde as variáveis são indexadas por $(i,j) \in [w] \times [s]$.

- $\text{Tribes}_{w,s}$ é trivialmente computável por um circuito FND de tamanho $s+1$.
- Toda árvore de decisão que computa $\text{Tribes}_{w,s}$ tem que ter profundidade ws — $\text{Tribes}_{w,s}$ é evasiva.

Tribes_n

Nós estamos mais interessados na seguinte escolha de parâmetros:

- Para cada $w \geq 1$, escolhemos s o maior inteiro tal que

$$(1 - 2^{-w})^s = \Pr[\text{Tribes}_{w,s}(x) = 0] \geq 1/2.$$

- $n = ws$.

Desta forma temos que Tribes_n é uma função "imparcial", os valores 1 e 0 aparece com basicamente a mesma probabilidade. Também temos que

- $s = \Theta(\frac{n}{\log n})$.
- $w = \log n - \log \log n - o(1)$.

Na verdade, $s \approx 2^w ln(2)$ e portanto $(1 - 2^{-w})^s \to 1/2$ com $w \to \infty$.

Teorema de Baker-Gill-Solovay

O teorema de Baker-Gill-Solovay diz que existem oráculos A e B tais que

- $\mathrm{P}^A = \mathrm{NP}^A$.
- $\mathrm{P}^B \neq \mathrm{NP}^B$.

Nós podemos provar que existe B tal que $\mathrm{P}^B \neq \mathrm{NP}^B$ (a parte não-trivial do teorema) usando o fato que a função Tribes_n é evasiva.

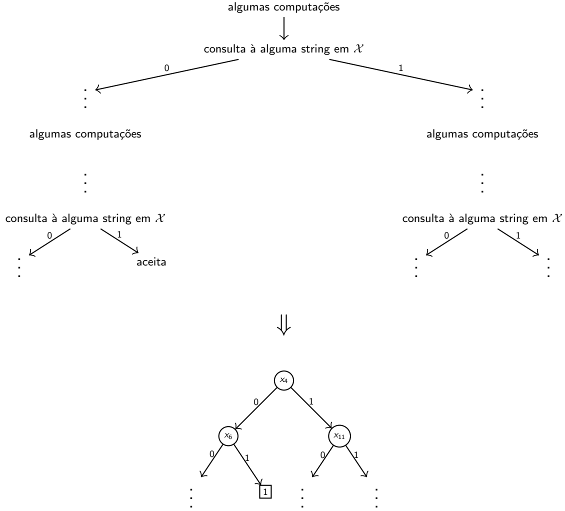
Teorema de Baker-Gill-Solovay - Prova

Seja M uma máquina de Turing de tempo polinomial que tem uma fita de oráculo e $x \in \{0,1\}^*$. Nós consideramos o seguinte:

- $\mathcal{X}$ um subconjunto finito de $\{0,1\}^*$.
- $A \subseteq \{0,1\}^* \setminus \mathcal{X}$ um oráculo.
- $T^{\mathcal{X}}_{M^A,x}$ uma árvore de decisão que recebe a string característica de um oráculo subconjunto de $\mathcal{X}$.
  - A string característica de $B \subseteq \mathcal{X}$ é a string x_B que é 1 no i-ésimo bit se a i-ésima string em $\mathcal{X}$ (sobre alguma enumeração das strings binárias) está em B.
- $T^{\mathcal{X}}_{M^A,x}(x_B) = 1 \iff M^{A \cup B}(x) = 1$.

Teorema de Baker-Gill-Solovay - Prova

Se M é uma máquina de Turing de tempo polinomial e $x \in \{0,1\}^*$.



Teorema de Baker-Gill-Solovay - Prova

No caso especial em que $\mathcal{X} = \{0,1\}^n$, $n = |x|$.

- $T_{M^A,x}^{\{0,1\}^n}$ tem profundidade polilogarítmica (o que é $\ll n$).
- Pois se M roda em tempo $\leq n^c$ então M faz no máximo n^c consultas ao oráculo.
- $n^c = \text{polylog}(2^n)$.

Teorema de Baker-Gill-Solovay - Prova

Nós consideramos a seguinte linguagem.

$$L(B) = \{1^n | \text{Tribes}_{n_w}(x_{B^{=n}}) = 1\}$$

- n_w satisfaz $2^{n-1} < n_w \leq 2^n$ (tal n_w é unico).
- $B^{=n} = B \cap \{0,1\}^n$.
- Como o tamanho da string $x_{B^{=n}}$ pode ser menor do que 2^n a entrada da função Tribes_{n_w} é na verdade a string $x_{B^{=n}}$ truncada.

Para todos $A \subseteq \{0,1\}^*$, $L(A) \in \text{NP}^A$.

- Dado um índice i que é múltiplo de s (o número de tribos) verifica se as strings $x^{(i+1)}, \ldots, x^{(i+w-1)}$ estão em A com $\Theta(\frac{n}{\log n})$ consultas.
  - $x^{(i)}$ é a i-ésima string de tamanho n na ordem lexicográfica.

Teorema de Baker-Gill-Solovay - Prova

$M_1, M_2, \ldots$ uma enumeração de máquinas de Turing de tempo polinomial e $p_1, p_2, \ldots$ seus tempos de execução. Escolha n_1 de forma que $p_1(n_1) < 2^{n_1}$ e $B(0) = \emptyset$.

- Primeiro estágio:
  - Defina B' de forma que $T^{\{0,1\}^{n_1}}_{M^{B(0)},1^{n_1}}(x_{B'}) \neq \text{Tribes}_{n_w}(x_{B'})$ (as funções Tribes são evasivas).
  - Faça $B(1) = B(0) \cup B'$.
- i-ésimo estágio:
  - Escolha n_i tal que $p_i(n_i) < 2^{n_i}$ e $n_i > n_{i-1}$.
  - B' tal que $T^{\{0,1\}^{n_i}}_{M^{B(i-1)},1^{n_i}}(x_{B'}) \neq \text{Tribes}_{n_w}(x_{B'})$
  - Faça $B(i) = B(i-1) \cup B'$.
- Por fim nós fazemos $B = \bigcup_{i \geq 1} B(i)$.

Teorema de Baker-Gill-Solovay - Prova

Então podemos argumentar que $\mathrm{P}^B \neq \mathrm{NP}^B$.

- Nós definimos cada $B(i)$ de forma que a máquina M_i falha em decidir $L(B)$ corretamente na entrada 1^{n_i} quando M_i tem acesso a $B(i)$.
- Como $B(i)$ é consistente com B, M_i deve falhar em decidir $L(B)$ corretamente na entrada 1^{n_i} com acesso a B.

$\square$

P $\neq$ NP para oráculos aleatórios

Além de Tribes$_n$ ser evasiva, ela nem mesmo pode ser aproximada por árvores de decisão com profundidade polilogarítmica.

- Nós dizemos que a árvore de decisão T aproxima uma função f se $T(x) = f(x)$ para quase todas as entradas.

Teorema

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

$$\Pr_{x \sim \{0,1\}^n}[A(x) = Tribes_n(x)] < 0,51$$

$\mathrm{P} \neq \mathrm{NP}$ para oráculos aleatórios

Bastar provar que qualquer árvore de consulta que faz consultas à somente uma fração constante das tribos não consegue aproximar Tribes_n.

- $g(x) = 1 \iff$
  pelo menos uma das primeiros $\frac{1}{100}$ das tribos é unanimamente 1.
- Temos que $\Pr[\mathrm{Tribes}_n(x) \neq g(x)] = \mathrm{E}[(\mathrm{Tribes}_n(x) - g(x))^2]$.
- Como $\mathrm{Tribes}_n(x) \geq g(x)$:
  - $\Pr[\mathrm{Tribes}_n(x) \neq g(x)] = \mathrm{E}[\mathrm{Tribes}_n(x) - g(x)]$.

$\mathsf{P} \neq \mathsf{NP}$ para oráculos aleatórios

Então temos que

$$\begin{aligned}
\Pr[\text{Tribes}_n \neq g(x)] &= \mathrm{E}[\text{Tribes}_n(x) - g(x)] \\
&= \mathrm{E}[\text{Tribes}_n(x)] - \mathrm{E}[g(x)] \\
&= \Pr[\text{Tribes}_n(x) = 1] - \Pr[g(x) = 1] \\
&= 1 - (1 - 2^{-w})^s - 1 + (1 - 2^{-w})^{\frac{1}{100}s} \\
&= (1 - 2^{-w})^{\frac{1}{100}s} - (1 - 2^{-w})^s.
\end{aligned}$$

Com w tendendo ao infinito isso é maior do que

$$2^{-1/100} - 1/2 - 0,001 > 0,492.$$

- Portanto,

$$\Pr[\text{Tribes}_n(x) = g(x)] < 1 - 0,492 = 0,508 < 0,51.$$

$\mathrm{P} \neq \mathrm{NP}$ para oráculos aleatórios - Lei 0-1 de Kolmogorov

- Para uma sequência $X_1, X_2, \ldots$ de variáveis aleatórias mutualmente independentes:
  - $G_n = \sigma\left(\bigcup_{i=n}^{\infty} X_i\right)$ é a menor σ-álgebra para qual cada $X_i, i \geq n$, é mensurável.
  - $\mathcal{T} = \bigcap_{n=1}^{\infty} G_n$ é a σ-álgebra caudal de $X_1, X_2, \ldots$.

Teorema

Seja $X_1, X_2, \ldots$ uma sequência de variáveis aleatórias independentes e $\mathcal{T}$ a σ-álgebra caudal destes eventos. Então todo evento $E \in \mathcal{T}$ satisfaz $\Pr[E] \in \{0, 1\}$.

Revisando a prova do teorema de Baker-Gill-Solovay

Para provar o teorema de Baker-Gill-Solovay nós utilizamos os seguintes passos:

- Máquina de Turing com acesso a um oráculo e uma string:
  - Árvore de decisão de profundidade polilogarítmica.
- Para cada oráculo $B \subseteq \{0,1\}^*$ nós definimos a linguagem unária $L(B)$ que está em NP^B.
  - Cada $L(B)$ está "estruturada" da mesma forma usando as funções Tribes_n.

Revisando a prova do teorema de Baker-Gill-Solovay

Para provar o teorema de Baker-Gill-Solovay nós utilizamos os seguintes passos (continuando):

- Nós definimos uma sequência $n_1, n_2, \ldots$, com $1 \leq n_1 < n_2 < \ldots$.
- Árvores de decisão de profundidade polilogarítmica não são capazes de computar as funções Tribes_n:
  - Um oráculo $B(i)$ em que a i-ésima máquina de Turing de tempo polinomial falha em decidir $L(B(i))$ para a string 1^{n_i} relativo a $B(i)$.
- E como último passo nós fazemos $B = \bigcup_{i \geq 1} B(i)$.

Revisando a prova do teorema de Baker-Gill-Solovay

Agora vamos considerar $\mathcal{C}_1$ e $\mathcal{C}_2$ classe de complexidades quaisquer e queremos provar que existe um oráculo $B \subseteq \{0,1\}^*$ tal que $\mathcal{C}_1^B \neq \mathcal{C}_2^B$.

- Um predicado P para uma linguagem em $\mathcal{C}_1$ e uma string x:
  - Representados por alguma classe $\mathcal{D}$ de "dispositivos computacionais".
  - Ou seja, $P(B,x) = 1 \iff D_x(B) = 1$, onde $D_x \in \mathcal{D}$.
- Para cada oráculo $B \subseteq \{0,1\}^*$ nós definimos a linguagem unária

$$L(B) = \{1^n | f(x_B) = 1\},$$

- para alguma função f que não pode ser computada em $\mathcal{D}$ e tal que $L(B) \in \mathcal{C}_2^B$.

Revisando a prova do teorema de Baker-Gill-Solovay

Agora vamos considerar $\mathcal{C}_1$ e $\mathcal{C}_2$ classe de complexidades quaisquer e queremos provar que existe um oráculo $B \subseteq \{0,1\}^*$ tal que $\mathcal{C}_1^B \neq \mathcal{C}_2^B$ (continuando).

- Daí podemos repetir o mesmo argumento que usamos para a prova do Teorema de Baker-Gill-Solovay.

Ou seja, podemos generalizar a prova do teorema de Baker-Gill-Solovay para provar outras separações por oráculo.

PH vs PSPACE relativizado

Lembrando a classe PH:

- Para todo $k \geq 1$, uma linguagem L está em Σ_k^p se e somente se

$$x \in L \iff \exists x_1 \forall \dots Q_k x_k M(x, x_1, \dots, x_k = 1,$$

  onde M é uma máquina de Turing que roda em tempo $p(n)$ e Q_k é $\exists$ se k é ímpar e $\forall$ se k é par.
  - Cada x_i tem tamanho no máximo $p(|x|)$.
- $\mathrm{PH} = \bigcup_{i \geq 1} \Sigma_k^p$.
- Vamos denotar a classe Σ_k^p com acesso à oráculo B por $\Sigma_k^{p,B}$.
- $\mathrm{PH}^B = \bigcup_{i \geq 1} \Sigma_k^{p,B}$.

PH vs PSPACE relativizado

Nós já sabemos que PH $\subseteq$ PSPACE. É verdade que existe $B \subseteq \{0,1\}^*$ tal que $\mathrm{PH}^B \not\supseteq \mathrm{PSPACE}^B$?

- Um predicado $P_{k,i}$ em Σ_k^p é

$$P_{k,i}(x) \iff \exists x_1 \forall \dots Q_k x_k M_i(x, x_1, \dots, x_k) = 1,$$

  onde M_i é a i-ésima máquina de Turing de tempo polinomial e Q_k é $\exists$ ou $\forall$ dependendo de k ser ímpar ou par, respectivamente.
- Podemos então enumerar todos os predicados P_i em PH.
- Para $x \in \{0,1\}^*$ e $B \subseteq \{0,1\}^*$:

$$P_{i,x}(B) = 1 \iff P_i^B(x) = 1,$$

  onde P_i^B é o predicado P_i com acesso ao oráculo B.

PH vs PSPACE relativizado

O primeiro passo para provar que existe um oráculo B tal que $\mathrm{PH}^B \neq \mathrm{PSPACE}^B$ é mostrar que $P_{i,x}$ pode ser representado por um circuito AC^0.

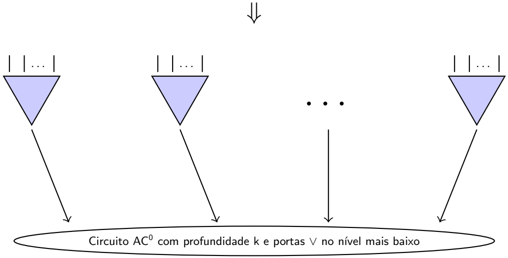
- Um predicado PH P_i é da forma

$$P_i(X) \iff \exists x_1 \forall x_2 \ldots Q_k x_k M_i(x, x_1, \ldots, x_k) = 1.$$

  - Cada quantificador $\exists \Rightarrow$ uma camada de portas $\vee$.
  - Cada quantificador $\forall \Rightarrow$ uma camada de portas $\wedge$.
- Representamos $P_{i,x} \equiv P_{k,i,x}$, para algum $k \geq 1$, por um circuito AC^0 de profundidade $k+1$.

PH vs PSPACE relativizado

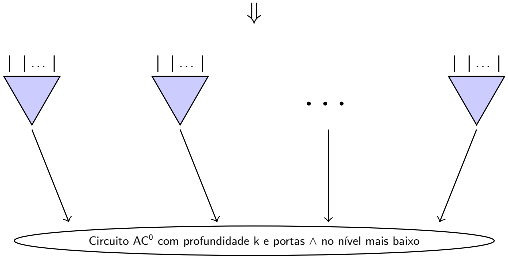
$$P_i(x) \iff \exists x_1 \forall \dots \exists_k x_k M_i(x, x_1, \dots, x_k) = 1$$



Cada árvore de decisão representa a computação de M_i com a entrada x e todas as possíveis escolhas de $x_1, x_2, \dots, x_k$.

PH vs PSPACE relativizado

$$P_i(x) \iff \exists x_1 \forall \ldots \forall_k x_k M_i(x, x_1, \ldots, x_k) = 1$$



Cada árvore de decisão representa a computação de M_i com a entrada x e todas as possíveis escolhas de $x_1, x_2, \ldots, x_k$.

PH vs PSPACE relativizado

Para um predicado $P_{i,x}$ nós temos um circuito $C_{P_{i,x}}$ com $N = 2^{\mathrm{poly}(|x|)}$ entradas que satisfaz:

- Tamanho $\Theta(2^k N)$ e profundidade $k + 1$.
- Fan-in $\mathcal{O}(\log N)$ nas portas lógicas no nível mais baixo e fan-in $\mathcal{O}(N)$ nas demais portas.

PH vs PSPACE relativizado

Consideramos a seguinte linguagem para cada $B \subseteq \{0,1\}^*$:

$$L(B) = \{1^n | \text{Parity}(B^{=n}) = 1\}$$

Nós temos que $\text{L}(B) \in \text{PSPACE}^B$. Nós temos o seguinte teorema:

Teorema

Seja $d > 0$ um inteiro. Para n suficientemente grande temos que qualquer circuito de profundidade d com fan-in polylog(n) *no teu 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.*

PH vs PSPACE relativizado

Teorema

Seja $d > 0$ um inteiro. Para n suficientemente grande temos que qualquer circuito de profundidade d com fan-in $\mathrm{polylog}(n)$ *no teu 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.*

- Como haviamos discutidos, isto prova que existe um oráculo $B \subseteq \{0,1\}^*$ tal que $\mathrm{PH}^B \neq \mathrm{PSPACE}^B$.

Separando a hierarquia polinomial

Para separar a hierarquia polinomia relativo a algum oráculo nós temos que mostar que existe um oráculo $B \subseteq \{0,1\}^*$ tal que para todo $k \geq 1$

$$\Sigma_{k-1}^{p,B} \neq \Sigma_{k}^{p,B}$$

- Nós temos que mostrar que um único oráculo separa várias classes.
- O nosso framework funciona para este caso?

Para cada $B \subseteq \{0,1\}^*$ e $k > 1$ temos a seguinte linguagem:

$$L(B,k) = \{1^n | f^{k+1,N}(B^{=n}) = 1\}$$

Onde $N = \left(\sqrt{2/(k+1)}2^n\right)^{\frac{1}{k}}$ e $f^{k,n}$ são as funções de Sipser:

Definição

A função de Sipser $f^{k,n}$ é definida da seguinte forma:

$$\bigvee_{i_k=1}^{\sqrt{\frac{n}{\log n}}} \bigwedge_{i_{k-1}=1}^{n} \cdots \bigwedge_{i_2=1}^{n} \bigvee_{i_1=1}^{\sqrt{\frac{1}{2}dn \log n}} x_{i_1,i_2,\ldots,i_k}, \text{ se } k \text{ é par.} \tag{1}$$

e

$$\bigvee_{i_k=1}^{\sqrt{\frac{n}{\log n}}} \bigwedge_{i_{k-1}=1}^{n} \cdots \bigvee_{i_2=1}^{n} \bigwedge_{i_1=1}^{\sqrt{\frac{1}{2}dn \log n}} x_{i_1,i_2,\ldots,i_k}, \text{ se } k \text{ é ímpar.} \tag{2}$$

Separando a hierarquia polinomial

E nós podemos ver que as funções $f^{k,d}$ têm profundidade k e tamanho

$$1 + \sum_{i=0}^{k-2} n^i \sqrt{\frac{n}{\log n}} = 1 + \left(\frac{n^{k-1} - 1}{n - 1}\right) \sqrt{\frac{n}{\log n}}$$

O número de variáveis de entrada de $f^{k,n}$ é

$$m = n^{k-2} \sqrt{\frac{n}{\log n}} \sqrt{\frac{1}{2} kn \log n} = n^{k-1} \sqrt{k/2}.$$

E daí vemos que o circuito para $f^{k,n}$ tem tamanho linear.

Separando a hierarquia polinomial

Nós também temos um limitante inferior para as funçõs $f^{k,n}$:

Teorema

Seja $k > 2$ e n suficientemente grande, qualquer circuito de tamanho $< 2^{\Theta\left(\sqrt{\frac{n}{k \log n}}\right)}$ e profundidade $k-1$ não computa a função $f^{k,n}$ corretamente em todas as entradas.

Separando a hierarquia polinomial

- Seja $P_1, P_2, \ldots$ uma enumeração de todos os predicados PH tal que cada i é em particular um predicado $\Sigma^p_{k_i}$, para algum $k_i \geq 1$.
- Seja $n_1, n_2, \ldots$ uma sequência de inteiros tal que:
  1. $C_{P_i, 1^{n_i}}$ tem tamanho polinomial e fan-in no nível mais baixo polilogarítmico.
  2. $1 < n_1 < n_2 < \ldots$.
- Para cada i nós podemos fazer P_i falhar em computar $L(B, k_i + 1)$ na entrada 1^{n_i} graças ao limitante inferior para as funções de Sipser.
- Isto é suficiente para construir um oráculo B que faz a hierarquia polinomial ser infinita.