In Lecture 2 we proved that computing $PARITY$ on n inputs requires unbounded fan-in depth d AND/OR circuits of size $2^{n^{\Omega(1/d)}}$, and that the same bound applies for computing $MAJORITY$ by AND/OR/PARITY circuits. When d is a fixed constant, the required circuit size for $PARITY$ grows exponentially in the input size. However, if we set d to equal $\log n/\log\log n$, this bound does not say anything and for a good reason: $PARITY$ on n bits *can* be computed by a circuit of depth $\log n/\log\log n$ and size $O(n)$, and so can $MAJORITY$.

What happens when the depth of the circuit becomes logarithmic in the input size? We will look at circuits of *bounded* fan-in, that is a circuit in which each gate takes in a constant number of inputs from previous layers. For such a circuit to compute interesting functions on n bits, its depth must be at least $\Omega(\log n)$, for otherwise it would be too small to examine all its inputs. To obtain the simplest "reasonable" model of such circuits we will think of the depth as growing at the rate of $K \log n$ for some constant K. Which functions require large circuits of this type?

Although there are many candidate examples, we do not know of a single "explicit" function that provably requires log-depth circuits of size that grows even super-linearly in n. Nevertheless, such circuits are quite interesting for the following reason. One of our motivations for studying restricted depth circuits was to understand parallel computation. Bounded fan-in circuits of logarithmic depth turn out to be equivalent to branching programs, a model of *sequential* computation.

To be concrete, we will assume that our bounded-depth circuits have fan-in 2. This is mostly for convenience and without much loss in generality:

Claim 1. *If a function can be computed by a circuit of size s, depth d, and gates of any type of fan-in at most c, then it can also be computed by a circuit of fan-in 2, size $2^{O(c)}s$, and depth $(c + \log c)d$.*

An unbounded fan-in AND/OR/PARITY circuit of size s and depth d be converted into a fan-in 2 circuit of size $O(s \log s)$ and depth $O(d \log s)$ after we replace each of the AND, OR, and PARITY gates by a complete binary tree of gates of fan-in 2 of the same type. In particular, the $PARITY$ function on n bits has a linear-size fan-in 2 circuit of depth $\log n$. The $MAJORITY$ function on n bits has a fan-in 2 circuit of size $O(n \log n)$ and depth $O(\log n)$: It recursively computes the sum $x_1 + \cdots + x_n$.

1 Branching programs

A branching program is a device with some small number of states. Before it starts its computation, the device decides how it is going to process its input: Maybe first it looks at the input bit x_5, then x_2, x_7, x_2 again, and so on. At each time step, it updates its state as a function of its current state and the input bit it was looking at. The device can use different update rules in different time steps.

Definition 2. An *(oblivious) branching program* on n inputs of width w and length ℓ consists of a sequence of input positions $k(1), \ldots, k(\ell) \in \{1, \ldots, n\}$ and transition functions $f_1, \ldots, f_\ell \colon [w] \times \{0,1\} \to [w]$.

The branching program of width w computes a function $f \colon \{0,1\}^n \times [w] \to [w]$ as follows: On input (x, s_0), ℓ steps of computation are performed, where in step t the state is updated from s_{t-1} to $s_t = f_t(s_{t-1}, x_{k(t-1)})$. The output is the value of s_ℓ. (If the function of interest is of the type $f \colon \{0,1\}^n \to \{0,1\}$, we restrict the number of states in the first and last layer to 1 and 2 respectively.)

Branching programs model sequential computation with a bounded amount of memory. When w is of the form 2^k, we can think of the memory as represented by a fixed set of registers $R_1, \ldots, R_k$ taking $0, 1$ values. The program then consists of ℓ "instructions $f_1, \ldots, f_\ell$, where each instruction is of the type "look up some input bit and update the registers depending on its value".

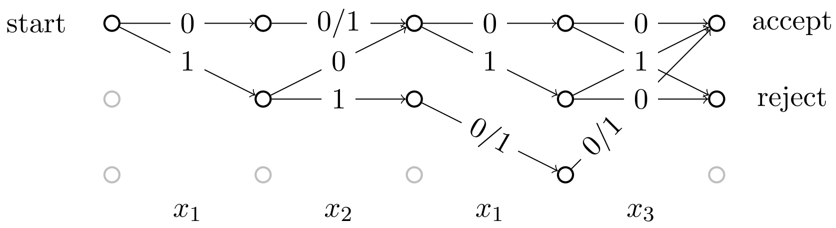

Figure 1: A width-3, length-4 branching program for (x_1 AND x_2) OR (x_1 XOR x_3).

The *PARITY* function can be computed by a width 2 branching program of length n: The input positions are $k(t) = t$ and the transition functions are $f_t(x_t, s) = s \oplus x_t$. A more natural way to describe this branching program is to say that it reads the input from left to right and maintains the parity of the input bits read so far in a single boolean-valued register. Similarly, the *MAJORITY* function can be computed by a width n branching program which reads the input from left to right, maintains the sum of the input bits read so far, and accepts if it exceeds $n/2$. Can *MAJORITY* be computed by a narrower branching program of reasonable length?

Let us start with small widths. An oblivious branching program of width two cannot even compute *MAJORITY* on 3 bits, regardless of its length.[1] In contrast, every function can be computed in width 3 and length $n2^n$ by the following claim:

Claim 3. *A DNF of size s and width w can be computed by an oblivious branching program of width 3 and length ws.*

Proof. The branching program has 3 states labeled 0, 1, and accept. The branching program reads the terms of the DNF in order and the variables within each term in order. The transitions can be chosen so that at any given point, the state is accept if at least one previously seen term has been satisfied, and otherwise to the current value of the current term. So the accept state is reached if and only if the input is a satisfying assignment to the DNF. □

In principle *MAJORITY* can be therefore computed by a width 3 oblivious branching program of exponential size. It is not known, but it is believed, that exponential size is necessary for this case. It is also not known what happens for width 4, but it turns out that *MAJORITY* has width 5 oblivious branching programs of size polynomial in its input!

2 Log-depth circuits and width-5 branching programs are the same

Barrington's theorem says that any small-depth circuit, in particular one for the *MAJORITY*, can be simulated by a branching program of width 5:

Theorem 4 (Barrington's theorem)**.** *If f has a depth d circuit of fan-in 2 then it has a branching program of width* 5 *and size* $2^{O(d)}$.

We will prove Barrington's theorem but with the constant 5 replaced by 8. Recall that a branching program of width 8 can be viewed as a machine with 3 registers taking values in $\{0, 1\}$. Let's call them A, B, and C.

Proof. Assume f has a circuit of depth d. We begin by changing the AND , OR , and XOR gates in the circuit into $\times$ and $+$ gates. These gates compute multiplication and addition over the binary field $\mathbb{F}_2$, respectively. We can represent any gate of fan-in 2 using $\times$ and $+$ gates and the following rules:

$$\overline{x} = 1 + x \qquad x \text{ XOR } y = x + y \qquad x \text{ AND } y = x \times y \qquad x \text{ OR } y = 1 + \overline{x} \times \overline{y}.$$

[1] The proof I came up with is not very insightful. I would be interested in finding some invariant of width-2 branching programs that can be proved inductively on the length and clearly excludes majority.

After this transformation, we obtain a circuit for f with × and + gates and depth $O(d)$. This circuit has an extra input that is fixed to the constant 1.

We now design a branching program for f. We will prove the following statement by induction on the depth d of f: There is a branching program of width 8 and size 4^d so that when the branching program starts with register contents A, B, and C, it ends its computation with register contents A, B, and $C + f(x_1, \dots, x_n)B$. The theorem then follows by initializing the registers to $A = 0$, $B = 1$, and $C = 0$.

We prove the inductive statement by looking at the top gate of f. If this gate is the constant 1 or a literal x_i or $\overline{x}_i$, then there f can be computed by a branching program of length 1. If $f = f_1 + f_2$, then we obtain a linear length branching program for g by combining the programs P_1 for f_1 and P_2 for f_2 like this:

$$(A, B, C) \xrightarrow{P_1} (A, B, C + f_1 B) \xrightarrow{P_2} (A, B, C + (f_1 + f_2)B)$$

By inductive hypothesis, each of P_1 and P_2 has length 4^{d-1}, so f has length $2 \cdot 4^{d-1} \leq 4^d$. If $f = f_1 \times f_2$, we combine P_1 and P_2 again as follows:

$$\begin{aligned}(A, B, C) \xrightarrow{P_1} (A + f_1 B, B, C) \xrightarrow{P_2} (A + f_1 B, B, C + f_2(A + f_1 B))& \\ \xrightarrow{P_1} (A, B, C + f_2 A + f_1 f_2 B) \xrightarrow{P_2} (A, B, C + f_1 f_2 B).&\end{aligned}$$

In each of the steps, the program P_1 or P_2 is applied but the registers are permuted in some order. Using the inductive hypothesis, f has length $4 \cdot 4^{d-1} = 4^d$, concluding the inductive argument. □

The converse of Barrington's theorem also holds. A *formula* is a circuit in which each gate has fan-out one.

Theorem 5. *If f has a branching program of width w and length ℓ then it has an AND/OR formula of depth* $(\log w + 1)(\log \ell)$.

Therefore the size of the shortest formula, the size of the smallest circuit of depth logarithmic in its size, the length of the shortest branching program of width 5, and the length of the shortest branching program of width 100 for the same function are all polynomially related.

Proof. Let $P \colon \{0, 1\}^n \times [w] \to [w]$ be the branching program for f. We give a formula for the function

$$\phi(s, t, x) = \begin{cases} 1, & \text{if on input } x, B \text{ goes from state } s \text{ to state } t \\ 0, & \text{otherwise.} \end{cases}$$

To construct ϕ, we split P in two parts P_1 and P_2 of equal length. Suppose we have already constructed formulas ϕ_1 and ϕ_2 for them. Then we write

$$\phi(s, t, x) = \mathrm{OR}_{u=1}^{w}(\phi_1(s, u, x) \text{ AND } \phi_2(u, t, x))$$

which describes the fact that if on input x, B goes from state s to state t, then it must do so thru some state u in the middle. The depth of ϕ is then bigger than the maximum depth of ϕ_1 and ϕ_2 by $\log w + 1$. Since ϕ_1 and ϕ_2 describe branching programs of half the length, we can continue the construction recursively and obtain a circuit of depth $(\log w + 1)(\log \ell)$ for B. (In the base case $\ell = 1$, ϕ depends on at most one input bit x_i so the formula has depth zero.) □

3 Streaming computation

A *read-once branching program* (or ordered binary decision diagram) is a branching program in which every input bit is read at most once. A *fixed-order* read-once branching program is one in which the inputs are read in the canonical order $x_1, x_2, \dots, x_n$. This is a model of streaming computation: At any point in time, the computation can only store a small amount of information about the "big data" stream $x_1, \dots, x_n$.

A fixed-order read-once branching program of width 2^n can compute any function $f\colon \{0,1\}^n \to \{0,1\}$, as its n state registers can remember the values of all n input bits read in order. The branching programs for $PARITY$ and $MAJORITY$ on n bits that we saw have width 2 and n, respectively. Here is an example of a function that requires a large branching program:

Claim 6. *The function $EQUAL(x, y) = (x_1 = y_1)$ AND $\cdots$ AND $(x_n = y_n)$ requires a read-once branching program of width 2^n under the ordering $x_1, \dots, x_n, y_1, \dots, y_n$.*

Proof. Let B be a branching program of width less than 2^n. Then there must be two distinct strings $x, x' \in \{0,1\}^n$ such that B reaches the same state on inputs x and x' in the first n steps of computation. For every $y \in \{0,1\}^n$, B must produce the same answer on inputs (x, y) and (x', y). But for $y = x$, $EQUAL(x, y)$ and $EQUAL(x', y)$ have different values, so B cannot compute $EQUAL$. □

Both the upper and lower bounds can be improved to show that a fixed-order read-once branching program of width $O(2^n/n)$ can compute any function on n bits, but there is an explicit function that requires programs of width $\Omega(2^n/n)$.

Instead of pursuing this direction, let us look at a more general type of streaming algorithm, one that makes several passes over its input stream. A fixed-order *read-k-times* branching program is a branching program of length nk that reads its variables in order $x_1, \dots, x_n$ k times in a row. When k is equal to n, such a branching program can emulate a read-once branching program without restriction on the order, and in particular it can compute the $EQUAL$ function on n input bits even in width 3. When k is much smaller, it is difficult to see how the additional passes over the input can help, so $EQUAL$ looks like a plausible hard function for this model. To prove it is, we will give a property that all small branching programs of this type have, but the $EQUAL$ function does not.

Theorem 7. *If $f\colon \{0,1\}^n \times \{0,1\}^m \to [w]$ is computed by a read-k-times branching program of width w in the order $x \in \{0,1\}^n$ followed by $y \in \{0,1\}^m$ then $\{0,1\}^n \times \{0,1\}^m$ can be partitioned into product sets $X_1 \times Y_1, \dots, X_{w^{2k}} \times Y_{w^{2k}}$, $X_s \subseteq \{0,1\}^n, Y_s \subseteq \{0,1\}^m$ such that f is a constant function on $X_s \times Y_s$ for all s.*

Proof. Let u_0 be the start state of the branching programs and v_i, u_i be the states immediately after reading the i-th copies of x and y, respectively. For each such sequence $s = (u_0, v_1, u_1, \dots, v_n)$ let Z_s be the set of inputs (x, y) for which the branching program visits this sequence of states in this order. Clearly the sets Z_s partition $\{0,1\}^{n+m}$, f is constant on each Z_s (as its value is determined by the final state) and there are w^{2k} possible sequences s (as the start state is fixed and at most w choices for every other state).

It remains to argue that each Z_s is of the form $X_s \times Y_s$. Let X_s consists of those inputs x that take the branching program from state u_0 to v_1 and from u_1 to v_2 and from u_2 to v_3 and so on. Let Y_s consist of those inputs y that take the branching program from v_1 to u_1 and from v_2 to u_2 and so on. Then Z_s consists precisely of those pairs (x, y) where $x \in X_s$ and $y \in Y_s$. □

Corollary 8. *If the EQUAL function can be computed by a read-k-times branching program of width w in the order $x_1, \dots, x_n, y_1, \dots, y_n$ then $w \geq 2^{n/2k}$.*

In particular, constant streaming algorithms for the $EQUAL$ function of this type require a linear number of passes over the input.

Proof. By Theorem 7, if a branching program of the desired type exist then there are sets X_s, Y_s with the stated properties such that $EQUAL(x, y)$ is constant on each $X_s \times Y_s$. Each such set can contain at most one input of the type $(x, y = x)$, because $EQUAL$ is not constant on any product set that contains two inputs of this form. It follows that the number of set-pairs w^{2k} must be at least as large as the number of inputs of the form (x, x) which equals 2^n. □

Randomized streaming algorithms

A more realistic model of streaming computation is one in which the algorithm has access to a sequence of random bits. Under this relaxation, the $EQUAL$ function becomes much easier to compute. For a $2n$ bit input, the algorithm chooses independent random strings $r_1, \dots, r_h$ in $\{0,1\}^n$ and accepts if and only if $IP(x, r_i) = IP(y, r_i)$ for all i between 1 and h. Here $IP(x, r)$ is the inner product modulo 2 function

$$IP(x, r) = \langle x, r \rangle = x_1 r_1 + \cdots + x_n r_n \mod 2.$$

If x is equal to y then the algorithm always accepts. If x is not equal to y the algorithm sometimes errs in its decision, but the probability it does so is quite small. The key insight is that if x and y are different, then the probability that $IP(x, r)$ and $IP(y, r)$ are equal is exactly $1/2$ over the choice of r. To see this, notice that $IP(x, r) + IP(y, r) = IP(x + y, r)$, so even after all bits of r are fixed except the one in which x and y differ, this expression is equally likely to be zero and one. After repeating this for h times the probability of making an error goes down to $1/2^h$.

This algorithm can be implemented by a fixed-order read-once branching program of width 2^{2h}: The algorithm only needs to track the $2h$ values $IP(x, r_1), IP(y, r_1), \dots, IP(x, r_h), IP(y, r_h)$, each of which is a parity in x or y and can be implemented with one bit of memory.

A *randomized branching program* is a branching program in which the output of every transition can also depend on the value of some random string $r \in \{0,1\}^*$. We say a randomized branching program B computes a function f with error at most ε if for every input x, $\Pr_r[B(x; r) \neq f(x)] \leq \varepsilon$, where $B(x; r)$ denotes the output of branching program B on input x and randomness r.

We just saw that the $EQUAL$ function can be computed by a fixed-order read-once randomized branching program of width 2^h with error 2^{-h} for any h. On the other hand, the IP function itself is hard for this model. To show this, we first generalize Theorem 7 to randomized branching programs.

Lemma 9. *If a randomized branching program computes f with error at most ε then for every distribution D on $\{0,1\}^n$ there exists a deterministic branching program B of the same width and the same read pattern such that $B(x)$ differs from $f(x)$ with probability at most ε when x is sampled from D.*

Proof. If for every x, $\Pr_r[B(x; r) \neq f(x)] \leq \varepsilon$, then by averaging for every distribution D on inputs, $\Pr_{x \sim D, r}[B(x; r) \neq f(x)] \leq \varepsilon$. There must then exist at least one r such that $\Pr_{x \sim D}[B(x; r) \neq f(x)] \leq \varepsilon$. If r is fixed, $B(x; r)$ becomes a deterministic branching program that computes f with error at most ε. □

Combining Theorem 7 and Lemma 9 we can prove:

Theorem 10. *If $f\colon \{0,1\}^n \times \{0,1\}^m \to [w]$ is computed by a randomized read-k-times branching program of width w with error ε in the order $x \in \{0,1\}^n$ followed by $y \in \{0,1\}^m$ then there exists subsets $X \subseteq \{0,1\}^n$ and $Y \subseteq \{0,1\}^m$ with $|X| \cdot |Y| \geq 2^{n+m}/2w^{2k}$ and a constant c such that $\Pr_{x \sim X, y \sim Y}[f(x, y) \neq c] \leq 2\varepsilon$.*

Proof. By Lemma 9, under the assumption there exists a deterministic branching program B of the given type that $B(x, y)$ differs from $f(x, y)$ with probability at most ε when x, y are chosen from the uniform distribution. By Theorem 7, $\{0,1\}^{n+m}$ can be partitioned into w^{2k} sets $X_s \times Y_s$ on which B is constant. The sets of size less than $2^{n+m}/2w^{2k}$ cover less than half the points of $\{0,1\}^{n+m}$. Conditioned on (x, y) falling into one of the other sets, $B(x, y)$ therefore differs from $f(x, y)$ with probability at most 2ε. So there exists at least one set $X_s \times Y_s$ that is both of size at least $2^{n+m}/2w^{2k}$ and such that $\Pr[f(x, y) \neq B(x, y)] \leq 2\varepsilon$. B is constant on $X_s \times Y_s$ and the theorem follows. □

On the other hand, the inner product function is far from constant on large product sets:

Theorem 11. *For every pair of sets $X, Y \subseteq \{0,1\}^n$, $\Pr[IP(x, y) = 0]$ and $\Pr[IP(x, y) = 1]$ are at most $\frac{1}{2} + \frac{1}{2}\sqrt{2^n/|X||Y|}$, where x and y are sampled independently and uniformly from X and Y, respectively.*

Combining Theorems 10 and 11, it follows that a randomized fixed-order read-k-times branching program can compute IP with error ε only if

$$\frac{1}{2} + \frac{1}{2}\sqrt{\frac{2^n}{2^{2n}/2w^{2k}}} \geq 1 - 2\varepsilon$$

which is equivalent to $w^{2k} \geq 2^{n-1}(1-4\varepsilon)^2$. If we plug in $\varepsilon = 1/8$, we obtain

Corollary 12. *Read-k-times randomized branching programs for IP with error at most $1/8$ require width $\Omega(2^{n/2k})$.*

4 Eigenvalues of symmetric matrices

To prove Theorem 11 it will be convenient to view the function $IP(x,y)$ as a 2^n by 2^n matrix whose rows are indexed by the x-strings, whose columns are indexed by the y-strings, and whose (x,y) entry has the value $IP(x,y)$. We will also represent the subsets X and Y from the theorem by the *indicator functions*

$$f(x) = \begin{cases} 1, & \text{if } x \in S \\ 0, & \text{otherwise} \end{cases} \qquad g(y) = \begin{cases} 1, & \text{if } y \in T \\ 0, & \text{otherwise} \end{cases}$$

which we can also interpret as 2^n-dimensional vectors obtained by listing all the values $f(x)$ (and $g(x)$) as x ranges over $\{0,1\}^n$. The utility of this representation comes from the following identity

$$\text{number of pairs } (x,y) \in X \times Y \text{ such that } IP(x,y) \text{ equals } 1 = \sum_{x,y\in\{0,1\}^n} f(x)g(y)IP(x,y).$$

Indeed the summation on the right picks up a 1 exactly for those inputs for which $x \in X$ and $y \in Y$ and $IP(x,y) = 1$. To turn this count into a probability we divide by the cardinality of the sample space:

$$\Pr_{x\sim X, y\sim Y}[IP(x,y) = 1] = \frac{1}{|X||Y|}\sum_{x,y\in\{0,1\}^n} f(x)g(y)IP(x,y). \tag{1}$$

To finish setting things up, for symmetry it will be convenient to represent the Boolean values 0 and 1 of IP by the real numbers -1 and 1, respectively. The resulting matrix is called the Walsh-Hadamard matrix

$$H(x,y) = 1 - 2IP(x,y) = (-1)^{\langle x,y\rangle}, \qquad \text{where } x,y \in \{0,1\}^n.$$

Substituting $IP = \frac{1}{2} + \frac{1}{2}H$ in (1) we get

$$\begin{aligned} \Pr[IP(x,y) = 1] &= \frac{1}{|X||Y|}\sum_{x,y\in\{0,1\}^n} f(x)g(y)\big(\tfrac{1}{2} + \tfrac{1}{2}H(x,y)\big) \\ &= \frac{1}{2|X||Y|}\sum_{x,y\in\{0,1\}^n} f(x)g(y) + \frac{1}{2|X||Y|}\sum_{x,y\in\{0,1\}^n} f(x)H(x,y)g(y) \\ &= \frac{1}{2} + \frac{1}{2|X||Y|}\sum_{x,y\in\{0,1\}^n} f(x)H(x,y)g(y). \end{aligned}$$

To understand the expression $\sum f(x)H(x,y)g(y)$ it will be helpful to think of it as the vector-matrix-vector product $f^\top Hg$. Here f stands for the 2^n-dimensional column vector whose x-th entry is $f(x)$; its entries are indexed by strings in $\{0,1\}^n$. Similarly, g is the 2^n-dimensional column vector whose y-th entry is $g(y)$, and H is the $2^n \times 2^n$ matrix whose (x,y)-th entry is $H(x,y)$. The value $f^\top Hg$ is related to the *spectral norm* of the matrix H. To explain this connection we review eigenvectors and eigenvalues. Every N by N symmetric matrix A is associated to a list of real eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N$ and a corresponding orthonormal basis of eigenvectors $f_1, f_2, \ldots, f_N$ satisfying $Af_z = \lambda_z f_z$.

Theorem 13. *For every symmetric matrix A*

$$|\lambda_1(A)| = \max_{f,g\neq 0} \frac{|f^\top Ag|}{\|f\|\|g\|}.$$

This quantity is called the spectral norm $\|A\|$ of A. Here $\|f\| = \sqrt{\sum_x f(x)^2}$ is the 2-norm of f.

Proof. If $\lambda_1 \geq 0$ we can set $f = g = f_1$ to get $f^\top Ag = f_1^\top Af_1 = \lambda_1(A)f_1^\top f_1 = \lambda_1(A)$. Therefore the right hand side is at least as large as $|\lambda_1|$.

To show it cannot be larger, we represent f and g as combinations $f = \sum_z \alpha_z f_z$ and $g = \sum_w \beta_w f_w$ of the eigenvectors. Then

$$f^\top Ag = \sum_{z,w} \alpha_z\beta_w f_z^\top Af_w = \sum_{z,w} \alpha_z\beta_w\lambda_w f_z^\top f_w = \sum_z \lambda_z\alpha_z\beta_z.$$

The last equality happens because the vectors f_z are orthonormal so all the terms with $z \neq w$ cancel out. Taking absolute values we get

$$|f^\top Ag| \leq \sum_z |\lambda_z||\alpha_z||\beta_z| \leq |\lambda_1| \cdot \sum_z |\alpha_z||\beta_z|.$$

The sum can be at most $\sqrt{\sum \alpha_z^2} \cdot \sqrt{\sum \beta_z^2}$ by the Cauchy-Schwarz inequality: A dot product is at most the product of the lengths. By Pythagoras' theorem this is equal to $\|f\| \cdot \|g\|$, so $|f^\top Ag| \leq |\lambda_1|\|f\|\|g\|$ and the right hand side is at most $|\lambda_1|$. □

Applying Theorem 13 to the matrix H we get that

$$\left|f^\top Ag\right| \leq |\lambda_1(H)|\|f\|\|g\| = |\lambda_1(H)|\sqrt{|X|}\sqrt{|Y|}$$

and therefore

$$\left|\Pr[IP(x,y) = 1] - \frac{1}{2}\right| \leq \frac{1}{2}\frac{|\lambda_1(H)|}{\sqrt{|X| \cdot |Y|}}$$

As $|\Pr[IP(x,y) = 1] - 1/2| = |\Pr[IP(x,y) = 0] - 1/2|$, to prove Theorem 11 it remains to bound the eigenvalues of H by $2^{n/2}$. This is where our choice of the inner product mod 2 function comes in handy: The rows of H are orthogonal. Moreover, since all entries are ± 1, each row has norm $2^{n/2}$. It follows that the square H^2 is 2^n times the identity matrix so all its eigenvalues are equal to 2^n. Since the eigenvalues of the square of a matrix A are the squares of the eigenvalues of A, all eigenvalues of H are $2^{n/2}$ or $-2^{n/2}$, so $|\lambda_1| = 2^{n/2}$ as desired.

References

The proof of Barrington's theorem presented here is due to Ben-Or and Cleve. The presentation borrows from lecture notes of Madhu Sudan. Our treatment of read-once and multiple-read restricted branching programs is based on a connection between branching programs and communication complexity. Some of the material is covered in the book *Communication Complexity* by Eyal Kushilevitz and Noam Nisan.