Barrington’s Theorem

In this lecture we present Barrington’s Theorem. We start with some motivation.

1 Branching programs

A branching program on the variable set $X = \{x_1, \ldots, x_n\}$ is a finite directed acyclic graph with one source node and sink nodes partitioned into two sets, Accept and Reject. Each non-sink node is labeled by a variable $x_i$ and has two outgoing edges labeled 0 and 1 respectively. An input $x \in \{0, 1\}^n$ is accepted if and only if it induces a chain of transitions that lead the start node to a node in Accept. The length of the program is the maximum length of any such path. We are only going to consider layered branching programs of length $\ell$. Here the nodes are partitioned into $\ell$ sets and edges only go from one layer to the next. The width of a layered branching program is the maximum number of vertices in any layer. The start node is in layer 1 and the sink nodes in layer $\ell$.

A branching program can be thought of as a space-bounded model of computation where space = log(width); from each state, we just look at 1 bit of the input. This is a clean model of space-bounded computation which abstracts from model-dependent Turing-machine issues such as keeping track of the position of the head on the input tape.

It is easy to see that AND : $\{0, 1\}^n \rightarrow \{0, 1\}$ can be computed by a branching program of width 2 and length $n + 1$. One can have similar branching programs for the parity function. However, it is not clear if, for example, the majority function can be computed by such branching programs. It can be shown that every function on $n$ bits can be computed by a branching program of width 3 and exponential length. It was conjectured that majority requires constant-width branching program of super-polynomial length $\ell \geq n^{\omega(1)}$.

In this lecture we present a surprising result by Barrington that in particular disproves this conjecture.

2 Barrington’s Theorem

Theorem 1 (Small depth $\Rightarrow$ short branching program). If $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is computable by a circuit of depth $d$, then $f$ is computable by a branching program of width 5 and length $\ell = 4^d$. In particular, if $d = O(\log n)$ then $\ell = \operatorname{poly}(n)$; in particular, majority is computable by a branching program of width 5 and polynomial length $\ell = n^{O(1)}$.

For the proof, we will construct a group program and then “convert” it into a branching program. Recall a group is a set of elements with an operation and inverses. We will be working with $S_5$, the group of permutations of 5 elements.
Definition 2. A group program of length \( \ell \) is \((g_0^1, \ldots, g_0^\ell), (g_1^1, \ldots, g_1^\ell), (k_1, \ldots, k_\ell) \) where for any \( i,j \): \( g_i^j \in S_5 \) and \( k_i \in \{1, \ldots, n\} \). We say that this program \( \alpha \)-computes \( f : \{0,1\}^n \to \{0,1\} \) if \( \forall x \),
\[
  f(x) = 1 \Rightarrow \prod_{i=1}^{\ell} g_i^{x_{k_i}} = \alpha
\]
\[
  f(x) = 0 \Rightarrow \prod_{i=1}^{\ell} g_i^{x_{k_i}} = 1_G;
\]
which we can write compactly as \( \forall x : \prod_{i=1}^{\ell} g_i^{x_{k_i}} = \alpha^{f(x)} \).

Abusing notation we say that a permutation \( g \in S_5 \) is a cycle if its graph consists of exactly one cycle of length 5. For example, \( 1 \to 5 \to 2 \to 3 \to 4 \to 1 \) is a cycle. We write it compactly as \((15234)\).

Theorem 3 (Small depth \( \Rightarrow \) short group program). Any function computable by circuit of depth \( d \) is \( \alpha \)-computed by a group program of length \( 4 \ell d \) for every cycle \( \alpha \).

Proof of Theorem 1 assuming Theorem 3. Let \( \alpha = (12345) \), consider the following branching program: nodes at layer \( i \) are labeled with \( x_{k_i} \), edges from layer \( i \) to layer \( i+1 \) labeled 0/1 are \( g_i^0/g_i^1 \). The start node is 1 and the accept node is 2. Then
\[
  f(x) = 1 \Rightarrow \prod_{i=1}^{\ell} g_i^{x_{k_i}} = (12345) \Rightarrow \text{start } \leadsto 2 \Rightarrow \text{accept}
\]
\[
  f(x) = 0 \Rightarrow \prod_{i=1}^{\ell} g_i^{x_{k+i}} = 1_G \Rightarrow \text{start } \leadsto 1 \Rightarrow \text{not accept}.
\]

\[\square\]

3 Proof of the Group Program Theorem 3

Lemma 4 (Does not matter what cycle you compute with.). Let \( \alpha, \beta \in S_5 \) be two cycles, let \( f : \{0,1\}^n \to \{0,1\} \). Then \( f \) is \( \alpha \)-computable with length \( \ell \) \( \iff \) \( f \) is \( \beta \)-computable with length \( \ell \).

Proof. First observe that \( \exists \rho \in S_5 \) such that \( \alpha = \rho^{-1} \beta \rho \). To see this let
\[
  \alpha = (\alpha_1, \alpha_2, \ldots, \alpha_5),
\]
\[
  \beta = (\beta_1, \beta_2, \ldots, \beta_5),
\]
\[
  \rho := (\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, \ldots, \alpha_5 \to \beta_5).
\]
Suppose that \((g_0^0, ..., g_0^{\ell})(g_1^1, ..., g_1^{\ell})(k_1, ... k_\ell)\) \(\beta\)-computes \(f\); we claim that \((\rho g_1^0, ..., \rho g_\ell^{\rho-1})(\rho g_1^1, ..., \rho g_\ell^{\rho-1})\) (with the same indices \(k_i\)) \(\alpha\)-computes \(f\). To see this, note that

\[
\prod_{i=1}^{\ell} g_i^{x_{k_i}} = 1 \Rightarrow \rho^{-1} \prod_{i=1}^{\ell} g_i^{x_{k_i}} = \rho^{-1} \cdot \rho = 1,
\]

and

\[
\prod_{i=1}^{\ell} g_i^{x_{k_i}} = \beta \Rightarrow \rho^{-1} \prod_{i=1}^{\ell} g_i^{x_{k_i}} = \rho^{-1} \beta \rho = \alpha.
\]

\[\square\]

**Lemma 5** \((f \Rightarrow 1 - f)\). If \(f : \{0, 1\}^n \rightarrow \{0, 1\}\) is \(\alpha\)-computable by a group program of length \(\ell\), so is \(1 - f\).

**Proof.** First apply the previous lemma to \(\alpha^{-1}\)-compute \(f\). Then multiply last group elements \(g_0^0\) and \(g_\ell^1\) in the group program by \(\alpha\).

\[\square\]

**Lemma 6** \((f, g \Rightarrow f \land g)\). If \(f\) is \(\alpha\)-computable with length \(\ell\) and \(g\) is \(\beta\) computable with length \(\ell\) then \((f \land g)\) is \((\alpha \beta \alpha^{-1} \beta^{-1})\)-computable with length \(4\ell\).

**Proof.** Concatenate 4 programs: (\(\alpha\)-computes \(f\), \(\beta\)-computes \(g\), \(\alpha^{-1}\)-computes \(f\), \(\beta^{-1}\)-computes \(g\)). \((f(x) = 1) \land (g(x) = 1) \Rightarrow\) concatenated program evaluates to \((\alpha \beta \alpha^{-1} \beta^{-1})\); but if either \(f(x) = 0\) or \(g(x) = 0\) then the concatenated program evaluates to 0. For example, if \(f(x) = 0\) and \(g(x) = 1\) then the concatenated program gives \(1 \cdot \beta \cdot 1 \cdot \beta^{-1} = 1\).

It only remains to see that we can apply the previous lemma while still computing with respect to a cycle.

**Lemma 7.** \(\exists \alpha, \beta\) cycles such that \(\alpha \beta \alpha^{-1} \beta^{-1}\) is a cycle.

**Proof.** Let \(\alpha := (12345), \beta := (13542),\) we can check \(\alpha \beta \alpha^{-1} \beta^{-1}\) is a cycle.

\[\square\]

**Proof of Theorem 3.** By induction on \(d\) using previous lemmas.

\[\square\]