IAS/PCMI Summer Session 2000
Clay Mathematics Undergraduate Program
Advanced Course on Computational Complexity

Lecture 14: Lower Bounds for Tree Resolution

David Mix Barrington and Alexis Maciel
August 3, 2000

1. Resolution and Tree Resolution

The *Resolution* proof system can be defined as a restriction of the Sequent Calculus that allows only variables to appear as formulas in the sequents and that uses only the cut rule. (This is not the usual definition of Resolution but it is equivalent to it.) What can we do with such limited proofs? One thing we cannot do is start with the axioms and build larger formulas because we cannot introduce connectives. In fact, with direct resolution proofs, we can essentially prove nothing else but the axioms.

The way to make Resolution useful is to use it as a refutation system. We start with the axioms and a set S of additional sequents and try to derive the empty sequent. If we succeed, then we have proved that S is *inconsistent* or *unsatisfiable*: no truth assignment can make all of the sequents in S true. Note that a sequent of the form $x_1, \ldots, x_m \rightarrow y_1, \ldots, y_n$ is equivalent to the disjunction (or clause) $\neg x_1 \vee \cdots \vee \neg x_m \vee y_1 \vee \cdots \vee y_n$. Therefore, refuting a set of sequents S is equivalent to refuting the CNF formula that consists of the conjunction of all the clauses in S.

This does not that mean that with Resolution we can only refute CNF formulas. We can view Resolution as a refutation system for general formulas if we first convert an arbitrary formula F to a CNF formula F_C by using the idea we presented in Basic Lecture 7 for reducing CIRCUIT-SAT to 3SAT. View the formula as a circuit, introduce new variables for the output of each of the gates, and define CNF formulas that express the fact that the output of each of the gates is correct. The conjunction of all these formulas together with the variable corresponding to the output gate is satisfiable if and only if the original formula is satisfiable. Therefore, refuting F_C also refutes F. In addition, the size of F_C is polynomial in the size of F. So if the size of a refutation of F_C is polynomial in the size of F_C, then it is also polynomial in the size of F.

In the previous lecture, we showed that the pigeonhole principle has polynomial-size Sequent Calculus proofs. In this lecture, we will prove an exponential lower bound on the size of a certain type of Resolution proofs of the pigeonhole principle. Of course, with Resolution, what we will consider is refuting the negation of the pigeonhole principle. More precisely, $\neg\text{PHP}_m^n$ corresponds to the following set of clauses:

$$\begin{array}{rcll} & \to & p_{i1}, \ldots, p_{im} & (\text{for } i \in [n]) \\ p_{ij}, p_{kj} & \to & & (\text{for } i, k \in [n], i < k, j \in [m]) \end{array}$$

As a simple example, consider refuting $\neg\text{PHP}_1^2$. The corresponding set of clauses is

$$\begin{array}{rcl} & \to & p_{11} \\ & \to & p_{21} \\ p_{11}, p_{21} & \to & \end{array}$$

These can be refuted by two applications of the cut rule. In fact, $\neg\text{PHP}_1^n$ can always be refuted by two cuts. More generally, if $n > m$, then $\neg\text{PHP}_m^n$ has a refutation of size $2^{O(m)}$. This implies that if $m = O(\log n)$, then $\neg\text{PHP}_m^n$ has a refutation of size $n^{O(1)}$, which is polynomial in the size of $\neg\text{PHP}_m^n$. In contrast, it is known that any Resolution refutation of $\neg\text{PHP}_m^n$ has size at least 2^m. When $m = n^{\Omega(1)}$, the lower bound is exponential.

In the next section, we will prove a similar lower bound of 2^m on the size of *tree-like* Resolution refutations of $\neg\text{PHP}_m^n$. To explain what a tree-like refutation (or proof) is consider a directed graph whose nodes are the lines of the proof. For each inference that occurs in the proof, connect the conclusion to the hypotheses. Viewing the last line of the proof at the top, that graph may or may not be a tree. If it is, then we say the proof is tree-like. For example, the simple refutation of $\neg\text{PHP}_1^2$ we mentioned earlier is tree-like. The restriction of resolution to tree-like refutations is called *Tree Resolution*.

In proving the lower bound, we will make use of *decision trees*. These are binary trees in which each node is labeled by some variable and the two edges coming out of a node are labeled 0 and 1. The leaves of the tree are each labeled by some output value. A decision tree computes a function as follows: given an input, start at the root and follow the edge corresponding to the value of the indicated input variable. Repeat this until, eventually, a leaf is reached. The output of the function on that input is the label of that leaf.

For example, a tree deciding whether the first bit of the input is equal to the last one would first query the first bit and then the last one. This tree is of depth two and

size four. To compute the AND function, simply query all the input bits in order. This gives a tree of depth n and size $O(n)$. To decide the PARITY language, simply query all the bits in order and explore all the possibilities. This produces a tree of depth n and size $2^{O(n)}$.

It is easy to see that the AND function cannot have smaller decisions trees since every bit must be queried. Can the PARITY language have trees of size $2^{o(n)}$? The answer is no. If a leaf is reached before n bits have been queried, than flipping one of the bits that was not queried would make the output of the tree incorrect. Therefore, every leaf must be reached after all n bits have been queried. But then, only one input string will reach each leaf so the tree must contain 2^n leaves, one for each input string.

2. A Lower Bound for Tree Resolution

In this section, we show that every tree-like Resolution refutation of $\neg\text{PHP}_m^n$ has size at least 2^m. This will be done in two steps.

First, if a set of clauses is unsatisfiable, then, for every truth assignment to the variables, at least one of the clauses will evaluate to false. The search problem associated with an unsatisfiable set of clauses is to find, given a truth assignment, one of these false clauses. We will show how to transform a tree-like refutation of $\neg\text{PHP}_m^n$ into a decision tree for the associated search problem. This tree will be of the same size as the proof. Then, our second step will be to show that this tree must be of exponential size.

So consider a tree-like refutation of $\neg\text{PHP}_m^n$. The decision tree will have exactly the same structure as the proof tree. Consider any line $\Gamma \to \Delta$ in the proof. We now show how to label the nodes and edges in the subtree rooted at that node so that this subtree solves the decision problem for all assignments that make the sequent false. To label the entire tree, apply this procedure to the node corresponding to the last line of the proof, which must be the empty sequent. Note that the empty sequent is false for every assignment, so the tree will work for every assignment.

If $\Gamma \to \Delta$ is not an axiom or a clause of $\neg\text{PHP}_m^n$, it must have been inferred by applying the cut rule to sequents of the form $\Gamma, x \to \Delta$ and $\Gamma \to x, \Delta$, where x is variable. The node corresponding to this sequent will be labeled by x. All the assignments that make $\Gamma \to \Delta$ false make all the variables in Γ true and all those in Δ false. If x is false for one of these assignments, then this assignment will make $\Gamma \to x, \Delta$ false so the edge going to that node will be labeled 0. If x is true for one of these assignments, then this assignment will make $\Gamma, x \to \Delta$ true so the edge going to

that node will be labeled 1. The trees rooted at these two nodes can now be labeled recursively so that they correctly solve their respective subproblems.

The base case occurs when $\Gamma \to \Delta$ is either an axiom or one of the clauses of $\neg\text{PHP}_m^n$. In this case, we are dealing with a leaf of the tree. Note that we only care about assignments that make $\Gamma \to \Delta$ false. If this sequent is an axiom, then there are no such assignments so we can label the corresponding leaf whatever we want. If the sequent is a clause of $\neg\text{PHP}_m^n$, then we label the leaf with that clause. This completes the construction of the decision tree.

Now we show that this tree must be of size 2^m. Each assignment to the variables describes a multivalued mapping of the n pigeons to the m holes. Among all these assignments, we will focus on those that correspond to a one-to-one mapping of a subset of m of the pigeons to the holes. We call these *critical truth assignments* (CTA's). We will also consider subsets of these assignments that correspond to CTA's in which particular pigeons are mapped to particular holes. For example, the set of all CTA's where pigeons 1, 2, 5 go to holes 3, 9, 1, respectively.

Suppose that a decision tree T correctly solves the search problem for all CTA's that have pigeons $i_1, \ldots, i_r$ going to holes $j_1, \ldots, j_r$, respectively. By induction on $m - r$, the number of holes still free, we show that the size of the tree must be at least 2^{m-r}. A lower bound on the size of the entire tree then follows from the case $m - r = m$, because the entire tree solves the problem for all assignments, including all CTA's.

The base case is when $m - r = 1$, which means that there is only one hole free. Since there are still two unassigned pigeons, this tree cannot be a single leaf because we still have no way of knowing which of the remaining pigeons will go to the remaining hole and which pigeon(s) will be left out. So the tree must be of size at least 3, which is no less than $2^{m-r} = 2$.

For the inductive step, suppose that $m - r > 1$ and that a decision tree correctly solves the search problem for all CTA's that have pigeons $i_1, \ldots, i_r$ going to holes $j_1, \ldots, j_r$, respectively. By the same argument used in the base case, this tree cannot simply be a leaf. The root of the tree then queries some variable x. If the CTA's under consideration either all have x true or all have x false, then follow the corresponding edge to the next node. Keep on going until you hit either a node that splits the CTA's or a leaf. Once again, that node cannot be a leaf.

Now suppose that this node is labeled p_{ij}. Since this variable splits the current set of CTA's in a nontrivial way, it must be that pigeon i is still unassigned and that hole j is still free. Consider the node that the edge labeled 1 leads to. Assign to that node the subset of the current CTA's that send pigeon i to hole j. The tree rooted

at that node solves the search problem for those assignments, so it must be of size at least 2^{m-r-1}.

Next, consider the node that the edge labeled 0 leads to. All the CTA's going there do not send pigeon i to hole j. Since there are at least two free holes, choose some other hole k and consider the subset of the current CTA's that send pigeon i to hole k. Assign that set of CTA's to that node. Once again, the tree rooted at that node solves the search problem for those assignments, so it must be of size at least 2^{m-r-1}. This implies that the size of the the original tree T must be at least twice that, which is 2^{m-r}, as desired. And this completes the proof of the lower bound.

3. Generalization to Depth-0.5 Sequent Calculus Proofs

All the formulas occurring in a Resolution proof are simple variables. Let us now consider relaxing this restriction a little by allowing some less limited formulas. For example, we could allow formulas that contain a single connective. These are formulas of depth 1 so we call the corresponding proof system *depth-1 Sequent Calculus*. Note that depth here refers to the depth of the formulas allowed in the proof, not to the depth of the graph or tree representing the proof. Resolution could be called depth-0 Sequent Calculus.

We can generalize this further and allow formulas of any constant depth d. This gives us constant-depth Sequent Calculus proofs, which are also called AC^0 proofs because polynomial-size, constant-depth formulas, when viewed as circuits, define precisely the class AC^0.

We know from the previous lecture that the general Sequent Calculus can prove PHP^n_m for any $m < n$ in polynomial size. We have now shown that Tree Resolution cannot refute $\neg\mathrm{PHP}^n_m$ in subexponential size if $m = n^{\Omega(1)}$. We also mentioned earlier that general Resolution cannot either.

What about constant-depth proofs? It turns out that they cannot prove PHP^n_m in subexponential size if $m = n - n^{o(1)}$. But they can prove PHP^n_m in quasipolynomial size if $m \leq n/2$. (Quasipolynomial size means size $n^{(\log n)^{O(1)}}$.) In fact, depth-1 proofs in which all the connectives have fan-in at most $(\log n)^{O(1)}$ are enough for this. We call this depth 0.5.

So when $m = n/2$, Resolution (and Tree Resolution) cannot refute $\neg\mathrm{PHP}^n_m$ in subexponential size but depth-0.5 proofs can do it in quasipolynomial size. What about tree-like, depth-0.5 proofs? The techniques in the proof of our lower bound

for Tree Resolution can be extended to show a subexponential lower bound for these proofs too. The construction of the decision tree must be modified to take into account the logical rules of the Sequent Calculus. The nodes in the decision tree will end up with labels corresponding to formulas involving $(\log n)^{O(1)}$ variables. So the lower bound argument will have to be modified accordingly. We leave the details as an exercise.

4. Exercises

1. Use the fact that $\neg\text{PHP}^n_{n-1}$ has Resolution refutations of size $2^{O(n)}$, to show that $\neg\text{PHP}^n_m$ has Resolution refutations of size polynomial in n if $m = O(\log n)$.

2. Extend the Tree Resolution lower bound argument to show that tree-like Sequent Calculus refutations of PHP^n_m have size at least $2^{m/t}$ if all the formulas in the refutation involve at most t variables. (t is not necessarily a constant, it can be a function of n.) Use this to show that tree-like, depth-0.5 proofs cannot refute $\neg\text{PHP}^n_m$ in subexponential size.