CS 2429 - Propositional Proof Complexity

Lecture #2: 19 September 2002

Lecturer: Toniann Pitassi

Scribe Notes by: Alex Hertel

1 Today's Topics:

- Proof Search (Automatization / Minimum Proof Length Problem)
- Proof Systems for Recognizing Tautologies and Unsatisfiability
- Proof System Hierarchy
- Standard Proof Systems
- Proof Length
- P-Simulations

2 Definitions: Proof Size versus Proof Search

As mentioned in the last lecture, the *size* of propositional proofs is very closely connected to complexity theory:

Theorem 1 *NP equals $coNP$ if and only if there exists a polynomially bounded propositional proof system.*

Today we will introduce a second concept that is equally important in studying proof complexity, namely the complexity of searching for a proof. To motivate this, consider a specific proof system such as DPLL for propositional unsatisfiability. This is a relevant example since this is the most popular proof system underlying automated theorem provers.

Assuming that NP does not equal $coNP$, it follows that there must be an infinite family of unsatisfiable CNF formulas that require superpolynomial-size DPLL proofs. (In fact, as we will see shortly, we can actually prove this lower bound unconditionally, without assuming anything about NP versus $coNP$.) On the other hand, there still are many examples of unsatisfiable formulas that do have efficient DPLL proofs, and then the question becomes one of proof search–how hard is it to *find* these short proofs. Ideally, we would like to develop an algorithm that takes as input an unsatisfiable CNF formula, and outputs a DPLL refutation of the formula, and such that the algorithm runs efficiently. Efficiently here means polynomial-time in the length of the shortest

DPLL proof of the formula. (Note that polynomial time in the size of the formula is unreasonable since DPLL is not polynomially bounded.) This motivates the following definitions, which introduce the concept of proof search complexity.

Definition Let V be a propositional proof system. The $MinimumProofLengthProblem$ for V: Given a boolean formula ϕ, $\phi \in$ TAUT, find the shortest V-proof of ϕ.

The above definition tries to find the absolute shortest proof. It turns out that for almost all proof systems, this problem is NP-hard. Therefore we relax the definition so that instead of the finding the optimal size proof, we want to come within a good approximation of the optimal.

Definition A proof system is *automatizable* if there is an algorithm A with input ϕ, $\phi \in$ TAUT with output V-proof of ϕ, and the running time of A is polynomial in $|\phi|$ + |shortest V-proof of ϕ|.

Fact There exists a polynomially-bounded automatizable proof system iff P=NP.

3 Proof System Hierarchy

We begin with a definition that allows us to compare two different propositional proof systems.

Definition A propositional proof system U p-simulates V if there exists a polynomial-time function, f such that $\forall$p$\forall$x V accepts (x, p) iff U accepts (x, f(p)). (This is analogous to a polynomial-time reduction.)

Definition Two propositional proof systems U and V are *polynomially equivalent* (or p-equivalent) if U p-simulates V and V p-simulates U.

The following diagram lists a number of standard propositional proof systems, and illustrates their position in the Proof System Hierarchy. An arrow from Proof System A to Proof System B indicates that A p-simulates B. Underneath the dashed line, a lack of an arrow between Proof Systems C and D indicates that C does not p-simulate D. Nothing should be inferred about the meaning of arrows that would cross the dashed line, but are not present.

Figure 1: Proof System Hierarchy

4 Resolution

Resolution is a proof system for UNSAT. It takes a CNF (Conjunctive Normal Form) formula, and proves that it is not satisfiable.

Resolution has only one inference rule:

$$\frac{(B \vee x) \quad (C \vee \bar{x})}{(B \vee C)}$$

In the above, B and C refer to a disjunction of literals, and x is a variable. A resolution refutation (proof) of f is a sequence of clauses C_1, C_2, ..., C_m such that $C_m = \phi$, where each C_i is either a clause in f, or follows from two previous clauses by the resolution rule of inference.

4.1 Example 1

The following diagram illustrates a Resolution Refutation in diagramatic tree form. The corresponding sequence of clauses can be obtained by taking the ordering of the clauses to be the ordering obtained in a breadth-first traversal of the tree. Note that the empty clause is referred to as the root, the clauses labelling the leaves of the tree are clauses from f, and clauses labelling internal nodes of the tree are clauses derived from previous clauses by the resolution rule.

Figure 2: Tree-like resolution refutation

Definition The *size* of a Resolution refutation is the number of clauses in the sequence.

Definition The *width* of a Resolution refutation R is defined as the number of literals in the largest clause of R.

Definition A Resolution refutation is $tree-like$ if the underlying directed acyclic graph is a tree.

5 Resolution Refinements

We mention some very common special cases of Resolution that come up in practice.

- A Regular Resolution is one in which no variable is resolved more than once on any path from the root to a leaf. The Resolution Refutation in Figure 2 is Regular.
- Tree Resolution is a diagramatic form of a resolution refutation. The underlying directed acyclic graph (DAG) of a Tree Resolution is a tree. Note that in a Tree Resolution, initial clauses may be reiterated multiple times, but derived clauses may only be used once. Figure 2 is an example of a Tree Resolution.
- Necessity can cause a single clause in a resolution proof to be appealed to multiple times. If this occurs, the proof's underlying topology is no longer tree-like. One can keep the topology tree-like by appealing to a separate copy of a clause each time it is needed again.
- DPLL is another form of Resolution that is polynomially-equivalent to tree-like Resolution. We will prove this next.

Theorem 2 *DPLL and tree-like Resolution are polynomially equivalent.*

Proof To prove this theorem, we need to show how to efficiently convert a DPLL tree for a given unsatisfiable formula f into a tree-like Resolution refutation of f, and conversely, how to convert a tree-like Resolution refutation of f into a DPLL-tree for f.

To construct a tree-like Resolution refutation from a DPLL tree, arrange the clauses to be resolved in the same order as the leaves of the DPLL tree. Then resolve with respect to the clauses and variables which the DPLL tree branched on last. This will construct a tree-like Resolution refutation with the same topology as the original DPLL tree.

To construct a DPLL tree from a tree-like Resolution refutation, label the branches of the resolution tree with the truth values which falsify the clauses at the leaves of the branches such that if some variable x was resolved at that level of the resolution tree then the branch leading to sub-trees whose leaves contain $\bar{x}$ should be labelled $x = 1$ and the branch leading to sub-trees whose leaves contain x should be labelled $x = 0$. This will construct a DPLL tree with the same topology as the original tree-like Resolution refutation. (There is a bit of subtlety in the case where the same variable is resolved twice along the same path. This would still produce a DPLL tree although some of the paths in the tree are unnecessarily long. Alternatively, one can show that in an optimal-size tree-like Resolution refutation, no variable is resolved more than once along any path.)

We will illustrate the above ideas with an example. We will use the same formula that we discussed earlier.

This DPLL Resolution Refutation is equivalent to that in Figure 2. Note that any path from the root to a leaf corresponds to a truth assignment which will falsify that leaf.

Figure 3: DPLL Resolution Refutation

5.1 Soundness and Completeness

From the above p-simulations, it is not hard to see that Resolution is sound and complete.

Theorem 3 *Resolution is sound and complete.*

Soundness is proved by induction on the size of the Resolution refutation, using the fact that the underlying resolution rule is sound. To prove completeness, we want to show that given an unsatisfiable CNF formula f, it has a Resolution refutation. Given f, derive a DPLL tree for f by querying the underlying variables for f is some fixed order. Since f is unsatisfiable, the DPLL tree obtained will be valid. Now by the p-simulation result, it follows that there is a (tree-like) Resolution refutation of f as desired.

6 Cutting Planes

Cutting Planes is a Proof System that is used to refute linear inequalities. A linear inequality is of the form $\Sigma_i a_i x_i \geq k$, where a_i, k are integers, and the underlying variables are x_i.

6.1 Axioms

1. $x_i \geq 0$
2. 1 - $x_i \geq 0$

6.2 Rules

1. Addition

   $\Sigma\ a_i x_i \geq k$, $\Sigma\ b_i x_i \geq k' \Rightarrow \Sigma\ (a_i + b_i) x_i \geq k + k'$

2. Multiplication

   $\Sigma\ a_i x_i \geq k \Rightarrow \Sigma\ c \cdot a_i x_i \geq c \cdot k$

3. Division

   $\Sigma\ 2 \cdot a_i x_i \geq k \Rightarrow \Sigma\ a_i x_i \geq \lceil \frac{k}{2} \rceil$

   For Example:

   $2x_1 + 2x_2 \geq 3 \Rightarrow x_1 + x_2 \geq 2$

A Cutting Planes refutation of a set of linear inequalities $L = \{C_1, C_2, \ldots, C_q\}$ is a sequence of inequalities, $S_1, S_2, \ldots, S_m$ where each S_i is either from L or is an axiom or follows from two previous inequalities by a valid rule, and the final inequality $S_m = (0 \geq 1)$.

Theorem 4 *Let L be the language corresponding to the set of all sets of linear inequalities with no integer solutions. Cutting Planes is sound and complete for L.*

Definition The *length* of a Cutting Planes proof is the number of lines in the proof.

Theorem 5 *Cutting Planes p-simulates Resolution for CNF formulas.*

Proof

1. Start with a CNF formula, f, and a Resolution derivation for f. As in example 1, let $f = (a \vee b \vee c)(a \vee \neg b)(\neg a \vee d)(\neg b)$, and consider the Resolution refutation for f shown in Figure 2.

2. Rewrite each clause in f as a set of linear inequalities by the following conversion. Given a clause $C = (l_1 \vee l_2 \vee \ldots \vee l_k)$ convert C to the inequality $f(l_1) + \ldots + f(l_k) \geq 1$, where $f(x) = x$ and $f(\neg x) \rightarrow 1 - x$.

3. Simulate the resolution steps by addition followed with division with rounding if required as shown in the following example for f.

Below is an example of the formula that we've been working with converted to a Cutting Planes refutation.

Figure 4: Cutting planes proof of the unsatisfiability of f