CS 2429 - Propositional Proof Complexity

Lecture #3: September 26, 2002

Lecturer: Toniann Pitassi

Scribe Notes by: Daniel Ivan

We have seen in the previous lectures different propositional proof systems, such as Resolution, Davis-Putnam, DPLL, and Cutting Planes. Today we will discuss Frege Systems and the Sequent Calculus (PK). We will then talk about some definitions and metrics which allows us to compare different proof systems.

1 The Sequent Calculus (PK)

It turns out that the Frege Systems and the Sequent Calculus are general, robust, and they are stronger than the other proof systems mentioned above.

We will briefly recall from the previous lectures that Frege Systems are based on an underlying set of complete connectives (for example $\neg$, $\vee$, $\wedge$), a finite set of schematic axioms which are tautologies, and rules of inference which allow to derive other tautologies by reasoning with arbitrary formulas.

The Sequent Calculus is based on statements of the form $A_1, A_2, \ldots, A_k \rightarrow B_1, B_2, \ldots, B_\ell$, meaning that the conjunctions of the A's implies the disjunction of the B's: $A_1 \wedge A_2 \wedge \ldots \wedge A_k \supset B_1 \vee B_2 \vee \ldots \vee B_\ell \Longleftrightarrow \overline{A}_1 \vee \overline{A}_2 \vee \ldots \vee \overline{A}_k \vee B_1 \vee B_2 \vee \ldots B_\ell$. The Sequent Calculus starts with an axiom schema of the type $A \rightarrow A$ which is valid, and for any sequents of formulas Γ and Δ, there are the following rules:

weakening left: $\dfrac{\Gamma \to \Delta}{\Gamma, A \to \Delta}$ **weakening right:** $\dfrac{\Gamma, \to \Delta}{\Gamma \to \Delta, A}$

exchange left: $\dfrac{\Gamma \to \Delta, A, B}{\Gamma \to \Delta, B, A}$ **exchange right:** $\dfrac{\Gamma, A, B \to \Delta}{\Gamma, B, A \to \Delta}$

contraction left: $\dfrac{\Gamma \to \Delta, A, A}{\Gamma \to \Delta, A}$ **contraction right:** $\dfrac{\Gamma, A, A \to \Delta}{\Gamma, A \to \Delta}$

$\neg$ **left:** $\dfrac{\Gamma \to \Delta, A}{\neg A, \Gamma \to \Delta}$ $\neg$ **right:** $\dfrac{A, \Gamma \to \Delta,}{\Gamma \to \Delta, \neg A}$

$\vee$ **left:** $\dfrac{A, \Gamma \to \Delta \quad B, \Gamma \to \Delta}{A \vee B, \Gamma \to \Delta}$ $\vee$ **right:** $\dfrac{\Gamma \to \Delta, A, B}{\Gamma \to \Delta, A \vee B}$

$\wedge$ **left:** $\dfrac{\Gamma, A, B \to \Delta}{\Gamma, A \wedge B \to \Delta}$ $\wedge$ **right:** $\dfrac{\Gamma \to \Delta, A \quad \Gamma \to \Delta, B}{\Gamma \to \Delta, A \wedge B}$

All of the above formulas satisfy the Subformula Property. The Subformula Property states that in any PK proof, any formula appearing in the proof must appear in the final sequent of the proof.

The following *cut rule* does not satisfy the Subformula Property, but it provides a way to shorten PK proofs.

cut rule: $\dfrac{\Gamma \to \Delta, A \quad A, \Gamma \to \Delta}{\Gamma \to \Delta}$

The **soundness** property of the PK proof systems says that *if some sequent formula is provable in PK, then it is valid.* The **completeness** property of the PK proof system says that *if some sequent formula is valid, then it is provable in PK.*

Theorem 1 *PK is sound and complete.*

Proof Soundness is proven by induction on the number of sequents in the proof. If the proof consists of a single sequent, then it must be an instance of the axiom schema, and therefore is valid. To prove the inductive step we consider each rule one at a time, and prove that the rules preserve soundness. That is, if the sequent S is obtained by a applying one of the rules to the two previous sequents S_1 and S_2, then if S_1 and S_2 are valid, then so is S.

To prove completeness, we want to show that if the sequent $\Gamma \to \Delta$ is valid, then it has a PK proof. We prove this by induction on the number of logical connectives occuring in $\Gamma \to \Delta$. If there are no connectives, and if it is valid, then it must be of the form $x_1, \ldots, x_k \to y_1, \ldots, y_l$, where some variable x occurs on both the left and the righthand side. Thus, this sequent can be obtained from the axiom $x \to x$ by weakening. For the inductive step, we will need the following inversion lemma.

Lemma 2 *(Inversion Lemma) Let S be a sequent derivable from two sequents S_1 and S_2 by one of the PK rules. Then if S is valid, so are S_1 and S_2.*

Now consider a sequent $\Gamma \rightarrow \Delta$ which involves k connectives. Choose some outermost connective occurring in either a formula from Γ or a formula from Δ. Apply the appropriate logical rule in reverse in order to obtain two sequents $S_1 = \Gamma_1 \rightarrow \Delta_1$, and $S_2 = \Gamma_2 \rightarrow \Delta_2$ such that $S = \Gamma \rightarrow \Delta$ can be obtained from these two sequents by applying the logical rule, and where both S_1 and S_2 contain at most $k-1$ logical connectives. By the inversion lemma, both S_1 and S_2 are valid because S is valid. Now by the induction hypothesis, both S_1 and S_2 have valid PK proofs. Notice that we actually proved that any valid sequent has a cut-free PK proof. That is, a PK proof where the cut rule is not used.

We define the *size* of a PK proof as the total number of connectives occurring in all the sequents of the proof. The *line size* represents the total number of sequents in the proof.

A PK proof has a tree-like structure when each intermediate sequent in the proof is used at most once. Otherwise, if there is any intermediate sequent in the proof which is used more than once, the PK proof is DAG-like (DAG stands for *Direct Acyclic Graph*).

2 Polynomial simulations

Now, as we have seen various proof systems, it would be interesting to compare different proof systems. We defined the notion of p-simulation earlier, which allowed us to compare the strength of two different proof systems for the *same* language. Now we will generalize this notion to allow us to compare proof systems for different languages. In what follows, L_1 and L_2 will be two different languages, usually with polynomial-time reductions between them (in both directions). Often both L_1 and L_2 will both be coNP-complete, but not always.

Definition Let $\mathcal{L}_1, \mathcal{L}_2$ be two languages (usually both co-NP complete), and let f_2 be a polynomial-time reduction from $\mathcal{L}_2$ to $\mathcal{L}_1$. Then V_1 for $\mathcal{L}_1$ polynomially simulates (p-simulates) V_2 for $\mathcal{L}_2$ under reduction f_2 if there exists a polynomial-time h such that $\forall\ x \in \mathcal{L}_2$, $V_2(x,p)$ accepts if and only if $V_1(f_2(x), h(p))$ accepts.

Notice that if V_1 and V_2 are two proof sytems proofs systems for the *same* language $\mathcal{L}$, then if V_1 polynomially simulates V_2, then lower bounds for V_1 imply lower bounds for V_2. To see this, consider some infinite collection of strings $x_1, \ldots,$ where all $x_i \in \mathcal{L}$ and $|x_i| > |x_{i-1}|$. Suppose that any V_1 proof of x_i requires superpolynomial size. Now suppose for sake of contradiction that each x_i has a polynomial-size V_1 proof, p_i. Then by the p-simulation, $h(p_i)$ will be a polynomial-size V_1 proof of x_i, a contradiction.

However, the above argument doesn't always work when V_1 and V_2 are proof systems for different languages, because the underlying reductions between them are not necessarily onto. Nonetheless, we can add an extra condition (essentially saying that the reduction can be efficiently proven correct within the proof system) allowing us to extract lower bounds for V_1 from lower bounds for V_2.

Lemma 3 *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two languages, and let f_1 be a polynomial-time reduction from $\mathcal{L}_1$ to $\mathcal{L}_2$ and let f_2 be a polynomial-time reduction from $\mathcal{L}_2$ to $\mathcal{L}_1$. Let V_1 be a proof system for $\mathcal{L}_1$ and let V_2 be a proof system for $\mathcal{L}_2$. Assume that: (1) V_1 polynomially-simulates V_2 under f_2; and (2) for all $x \in \mathcal{L}_1$, if $f_2(f_1(x))$ has a short V_1 proof, then so does x. Then lower bounds for V_1 imply lower bounds bounds for V_2, under reductions f_1 and f_2.*

Let $x_1, \ldots,$ be an infinite collection of strings where $x_i \in \mathcal{L}_1$ for all i, and $|x_i| > |x_{i-1}|$. Suppose that any V_1 proof of x_i requires superpolynomial size. Now suppose for sake of contradiction that $f_1(x_i)$ has a polynomial-size V_2 proof, p_i. Then by (1) $h(p_i)$ will be a polynomial-size V_1 proof of $f_2(f_1(x_i))$. Now by (2), there is also a short V_1 proof of x_i, contradicting the fact that x_i was supposed to require a large V_1 proof.

Definition Two proof systems V_1 for $\mathcal{L}_1$ and V_2 for $\mathcal{L}_2$ are said to be polynomially equivalent under reductions f_1 and f_2 if lower bounds for V_1 follow from lower bounds for V_2 and vice versa (for reductions f_1 and f_2).

There are surprisingly few examples comparing proof systems for different languages. One example is the following.

Theorem 4 *(Pitassi, Urquhart) The Hajos calculus and Extended Frege are polynomially equivalent*

The above theorem is interesting since it had been an open problem to prove lower bounds for the Hajos calculus. While we still do not know how to prove such lower bounds, the above theorem tells us that the problem is basically equivalent to proving lower bounds for Extended Frege systems! Moreover, formal relationships between different proof systems for different languages can help us to leverage ideas and analyses from one domain area to a different domain area.

3 Resolution

We will now begin to study Resolution in detail. Resolution is the most popular theorem prover known. It is the basis for many automated theorem provers for both propositional as well as first order logic. It is also the basis for the best current algorithms for satisfiability testing.

Before we begin to talk about what is hard for Resolution, we want to point out that there are some important special cases of CNF formulas that are easy for Resolution. The resolution system can produce polynomial-size proofs for any unsatisfiable 2CNF formula, and for any unsatisfiable Horn formula. To see this for 2CNF, notice that if we begin with a 2CNF formula, we can only generate clauses containing at most 2 literals. The number of such clauses is polynomial in n. Thus Resolution when run on a 2CNF formula will either produce all clauses of size 2, including the empty clause, and then halt (running in polynomial time) and output unsatisfiable, or the algorithm will get to a point where it will not be able to generate any new clause of size at most 2, and in this case if the empty clause has not yet been generated, then the algorithm can halt and output "satisfiable". In either case, the algorithm halts in polynomial-time. The case of Horn formulas will be left as a homework exercise.

We will now show that Resolution can be simulated by PK. After that, we will see that PK is stronger, by proving that the propositional pigeonhole principle has short PK proofs, but requires exponential-size Resolution proofs.

Theorem 5 *The PK proof system p-simulates the resolution proof system.*

Proof

First note that the PK proof system proves that a sequent is a tautology, whereas the Resolution proof systems shows that a set of clauses are unsatisfiable. But a set of clauses is unsatisfiable if and only if its negation is a tautology.

The idea of the proof is to associate a sequent formula to each clause of the refutation of the resolution prove. Each refutation rule applied to two clauses corresponds to a PK rule. Therefore the last clause of the resolution refutation will be equivalent to the sequent $\rightarrow$ which is unsatisfiable. Therefore if the resolution refutation prove refutes some set of clauses g, then by the correspondences described above between each clause of the refutation and a sequent in the PK proof $\neg g$ is proved to be unsatisfiable in PK.

Example Let $g = (x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (x_1 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee \overline{x}_2) \wedge (\overline{x}_3)$ If g has a resolution refutation, then $\neg g = (\overline{x}_1 \wedge \overline{x}_2) \vee (x_1 \wedge \overline{x}_3) \vee (\overline{x}_1 \wedge x_2) \vee (x_1 \wedge x_2) \vee (x_3)$ is polynomially (in the size of the refutation of g) provable in PK to be valid, which means $\rightarrow \neg g$ has a valid PK proof. So $g \rightarrow$ has a valid PK proof.

In order to carry out this simulation, we will convert each clause of g to an equivalent sequent.

In the resolution refutation, from the first two clauses of g: $(x_1 \vee x_2)$, $(\overline{x}_1 \vee x_3)$ we refute another clause $(x_2 \vee x_3)$. But notice that $(x_1 \vee x_2) \Longleftrightarrow \rightarrow x_1, x_2$, and $(\overline{x}_1 \vee x_3) \Longleftrightarrow \rightarrow \overline{x}_1, x_3$. The refutation $(x_2 \vee x_3) \Longleftrightarrow \rightarrow x_2, x_3$ is the bottom sequent of the other two mentioned above in the cut rule of the PK prove system. Therefore generalizing along all the clauses obtained by refutation in the resolution prove, every refutation rule is equivalent with a cut rule in the PK prove. For each literal we get rid of in the resolution rule, there is an equivalent cut rule in the PK prove by which we get rid of the same literal. The final step in the resolution rule is when we have an atom P and its negation $\neg P$ from which we refute the empty set. This corresponds to an axiom in the PK prove $\emptyset \rightarrow$ which is obviously valid. Therefore the PK prove of $g \rightarrow$ was done in time linear in the size of the resolution prove, and it follows that $\neg g$ is a tautology.

Our canonical example that is hard for Resolution is the propositional pigeonhole principle.

3.1 The Pigeon Hole Principle

The pigeon principle PHP_n^m says that there is no 1-1 function from m objects ('pigeons') to n objects ('holes') if $m > n$. The *onto* version of this, $ontoPHP_n^m$, says that there is no 1-1, onto function mapping m pigeons to n holes for $m > n$. This can be easily encoded as an unsatisfiable propositional formula over variables P_{ij} which represent pigeon i mapping to hole j. The clauses ensure that any satisfying assignment to these variables corresponds to a valid 1-1, onto function from m pigeons to n holes. There are four kinds of clauses:

- f is total: $(P_{i1} \vee P_{i2} \vee \ldots \vee P_{in})$, for $i = 1, \ldots, m$
- f is 1-1: $(\neg P_{ij} \vee \neg P_{kj})$, for $1 \leq i < k \leq m$, $j = 1, \ldots, n$
- f is onto: $(P_{1j} \vee P_{2j} \vee \ldots \vee P_{mj})$, for $j = 1, \ldots, n$
- f is a function: $(\neg P_{ij} \vee \neg P_{ik})$, for $i = 1, \ldots, m$, $1 \leq j < k \leq n$

Note: We usually leave out the function clauses. One can derive the relational form from the functional form by setting $P'_{ij} = P_{ij} \wedge \neg P_{i1} \wedge \ldots \wedge \neg P_{i(j-1)}$.

While PHP_n^{n-1} can be efficiently proven in EF or Cutting Planes proof systems, any Resolution proof must be exponentially large.

Theorem 6 *Any resolution proof of PHP_n^{n-1} requires size at least $2^{n/20}$.*

The idea is based on the *bottleneck counting.* One views truth assignments as flowing through the proof. Assignments start at $\emptyset$ and flow out toward input clauses. A clause in the proof only allows the assignments it falsifies to flow through it. A key property is that at a middle level in the proof, clauses must talk about lots of pigeons. Such a middle level clause falsifies only a few assignments and thus there must be many such clauses to let all the assignments flow through.

We say a truth assignment is *i-critical* if it matches all $n-1$ holes to all pigeons but pigeon i. Such an assignment is barely unsatisfying – it always satisfies all 1-1, onto and function clauses and all but one of the clauses saying that f is total. The only clause it falsifies is $C_i = (P_{i1} \vee P_{i2} \vee \ldots \vee P_{i(n-1)})$ which says that pigeon i is mapped somewhere.

The properties of critical truth assignments make it convenient to convert each such clause C to a positive clause $M(C)$ that is satisfied by precisely the same set of critical assignments as C. More precisely, to produce $M(C)$, we replace $\neg P_{ij}$ in C with $(P_{1j} \vee \ldots \vee P_{(i-1)j} \vee P_{(i+1)j} \vee \ldots \vee P_{nj})$.

The above translation has the important property that it preserves flow of critical assignments. More precisely, let P be a Resolution proof, and let $M(P)$ be the corresponding sequence of monotone clauses. Then for any critical truth assignment α and for any clause C in P and corresponding clause $M(C)$ in $M(P)$, α satisfies C if and only if α satisfies $M(C)$.