CS 2429 - Propositional Proof Complexity

Lecture #1: 12 September 2002

Lecturer: Toniann Pitassi

Scribe Notes by: Toniann Pitassi

1 Introduction to Propositional Logic

What is a proof system? In this course we will define a proof system in a very general way, as folows.

Definition A proof system for a language L is a polynomial time algorithm V such that for all x $x \in L$ if and only if there exists a string p such that $V(x,p)$ accepts.

In this definition, V is the verifier and the string p is an encoding of a proof that x is in L. The "if and only if" in the above definition encorporates the standard notions of soundness and completeness. Soundness (the "if" direction) means that if $V(x,p)$ accepts for some p, then x is in L. Completeness (the "only if" direction) means that if x is in L, then there is some string p such that $V(x,p)$ accepts.

Definition The complexity of V is a function f_V from natural numbers to natural numbers, defined as follows. $f_V(n)$ is

$$\max_{x \in L, |x|=n} \quad \min_{p \ s.t. \ V(x,p) accepts} |p|.$$

V is *polynomially bounded* iff f_V is a polynomial function of n.

We will now give several examples of proof systems, each for different languages L.

1.1 Example 1 Satisfiability

The canonical language with a polynomially bounded proof system is SAT. SAT consists of the set of all satisfiable formulas. For example, the following conjunctive normal form formula is in SAT:

$$f = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee \neg x_4) \wedge (x_1 \vee x_3 \vee \neg x_4) \wedge (x_3 \vee x_4).$$

The verifier V for SAT takes two strings, f and p as input, where f an encoding of a boolean formula, and p a truth assignment to the underlying variables of f. $V(f,p)$ accepts if and only if p is a satisfying assignment to f. Clearly V is polynomial-time computable for all inputs, and f is in SAT if and only if there exists a string p such that $V(f,p)$ accepts.

1.2 Example 2 DPLL

The most prominent and well studied proof systems are all for the languages $TAUT$ and $UNSAT$. $TAUT$ is the set of all boolean formulas that are tautologies, and $UNSAT$ is the complement of SAT, consisting of the set of all boolean formulas that are unsatisfiable. For example, the CNF formula $f = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2)$ is in $UNSAT$ because it is unsatisfiable.

Since $UNSAT$ and $TAUT$ are so important, we will explicitly define a proof system for them.

Definition A propositional proof system (pps) is a proof system for the set $UNSAT$ ($TAUT$) of propositional logic unsatisfiable (tautological) formulas.

The following simple theorem, giving the basic connection between proof system complexity and complexity thoery, was stated and proven by Cook and Reckhow in their seminal paper on propositional proof complexity.

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

The most well-studied proof system for $UNSAT$ is the DPLL system, which is both a proof system as well as a family of algorithms for SAT. Consider the example $f = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee x_3) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1)$. A DPLL proof is a tree with fanin 2 and with the following properties. Each internal node of the tree is labelled with an underlying variable x_i. If a node is labelled by variable x_i, then the two outgoing edges of the node are labelled by $x_i = 0$ and $x_i = 1$ respectively. Corresponding to each internal node is a formula, which is the value of the original formula under the partial evaluation of variables defined by the path to that node. For example, if a node in the tree is reachable by a path from the root labelled by $x_1 = 0$ and $x_2 = 0$, then the formula corresponding to this node is $(\neg x_3) \wedge (x_3)$. For each leaf node l in the tree, the formula corresponding to l must be the false formula. That is, for each leaf node l in the tree, the partial assignment corresponding to the path to l must force the formula f to false. Each leaf node is labelled by a clause C from the original f that is forced to false along this path.

The following figure illustrates a DPLL tree for the formula $f = (a \vee b \vee c)(a \vee \neg c)(\neg a \vee d)(\neg d \vee b)(\neg b)$.

The above proof system fits into our definition of a propositional proof system where the verifier interprets the second input p as the encoding of a tree as defined above, and checks that all of the relevant conditions are satisfied. The proof system is sound and complete since a formula will be unsatisfiable if and only if there exists a decision tree as defined above.

The DPLL algorithm for satisfiability associated with the above proof system starts with a formula f and tries to build a DPLL tree for f in a depth-first manner. Either the algorithm is successful and a valid tree is found, in which case the algorithm can output "unsatisfiable" or the algorithm gets to a point where a partial assignment is found that satisfies the formula, and in this case the algorithm can not only output "satisfiable", but can also output a satisfying assignment.

1.3 Example 3 The Hajos Calculus

Our next example is a proof system for graph non-q-colorability. An undirected graph $G = (V, E)$ is 3-colorable if there is a function mapping each of the vertices in V to one of three colors (Red, Blue or Green) and such that every two adjacent vertices are assigned distinct colors. The language 3COL is the set of all 3-colorable graphs. This problem is well-known to be NP-complete, and deciding non-3-colorability is therefore coNP-complete. The language qCOL can be defined similarly, and again is NP-complete as long as q is at least 3. In the 1960's in an attempt to prove the four color theorem, Hajos defined a proof system, now called the Hajos calculus, for deriving the set of all non-q-colorable graphs, for $q \geq 3$. The only axiom is the graph K_q, the complete graph on q vertices. There are three rules for building larger graphs as follows, the add, contract and join rules. (i) (Add) From G, can derive G' where G' is G with extra vertices or edges added to G; (ii) (Contract) From G, can derive G' where G' is G but with two nonadjacent vertices contracted into a single new vertex; (iii) (Join) Let G_i contain two vertices a_i and b_i and such that there is an edge from a_i to b_i. Then from G_1 and G_1, can derive G' where G' is the union G_1 and G_2 but with a_1 and a_2 contracted into a single new vertex, a, and with the edges (a, b_1) and (a, b_2) removed, and with the edge (b_1, b_2) added.

A Hajos calculus derivation that a graph G is not q-colorable is a sequence of graphs, $G_1, G_2, \ldots, G_m$ such that $G_m = G$, and all other graphs are either instances of K_q, or follow from one or two previously derived graphs by one of the above three rules. The size of a derivation is the sum of the sizes of all graphs in the derivation. Hajos proved in 1961 that a graph G is non-q-colorable if and only if it has a Hajos derivation. This fits into our standard definition of a proof system where $V(G, p)$ views p as the encoding of a Hajos calculs derivation of G.

It is still an open problem whether or not the Hajos calculus proof system is polynoimally bounded; that is, whether there exist non-3-colorable graphs that require superpolynomial-size Hajos derivations.

1.4 Example 4 Tree proofs for Hypergraph Transversal

Our next example is a proof system for a language that is not known to be NP-complete (and is probably not NP-complete). The hypergraph transversal problem is defined as follows. The input is a pair of set systems, $S = \{s_1, \ldots, s_k\}$ and $T = \{t_1, \ldots, t_l\}$ over an underlying universe $U = \{x_1, \ldots, x_n\}$. A set s is a minimal transversal for a set system T if every set in T has nonempty intersection with s, and furthermore s is minimal (removing any element from s will violate the first condition). The input is accepted if and only if S is the set of all minimal transversals of T. Note that the definition is symmetric: S is the set of all minimal transversals of T if and only if T is the set of all minimal transversals of S. This problem is not known to be in P, or known to be NP-complete. However, there is a quasipolynomial-time algorithm that solves it. It is easily seen to be equivalent to the monotone CNF/DNF equivalence problem: Given a monotone CNF formula f and a monotone DNF formula g, determine whether or not they are equivalent.

A common and natural proof for this problem is a transversal tree defined as follows. The internal nodes of the tree are labelled with underlying elements from U. If a node is labelled x_i, then the two edges out of x_i are labelled by $x_i = 0$ and $x_i = 1$ respectively. There is a pair of set systems associated with each node as follows. The original set system (S, T, and U) corresponds to the root. If a node in the tree has a corresponding path with the label $x_1 = 0$ then the subtree rooted at this node is labelled with the set system S', T' and U' where U' is U minus x_1, S'

consists of each set s of S, but with x_1 removed from s, and T' are those sets in T that do not contain x_1. In other words, this subtree is computing the transversals of S that do not contain x_1. Similarly, if a node has path with the label $x_1 = 1$, then the subtree rooted at this node is labelled by the set system S', T' and U' where again U' is U minus x_1, but now S' consists of those sets of S not containing x_1, and T' consists of those sets of T that contain x_1 but with x_1 removed. Finally, the leaves of the transversal tree should have both S' and T' empty. The proof system, $V((S,T,U),p)$ interprets p as a transversal tree and checks whether or not it is a valid tree for the triple (S,T,U). It is not known whether or not this proof system (for the hypergraph transversal language) is polynomially bounded.

2 Motivations, connections

It is striking that even when talking about totally different languages, that the natural proof systems for them often are strikingly similar. For example, the DPLL proof system is very similar to the proof system that we described for the transversal problem.

We will focus our attention on propositional proof systems for most of this course, although we will try to encorporate results about proof systems for other languages as we go along.

There are several motivations for studying proof complexity. First, it gives an interesting way to classify algorithms for solving SAT. Then once we obtain lower bounds for a particular proof system, it implies lower bounds for the corresponding class of algorithms for SAT. This can be viewed as an alternative to the circuit complexity approach for proving that P is different from NP. In the circuit approach, one tries to develop lower bounds for restricted circuit classes (for example, bounded-depth circuits, bounded-depth threshold formulas, etc) in the hopes of eventually building up enough machinery to prove that SAT does not have polynomial-size circuits.

An alternative approach based on proof complexity is to try to prove lower bounds for restricted classes of propositional proof sytsems. An advantage of this alternative approach is that the one can actually show that certain algorithmic methods that are actually used and studied for solving SAT, cannot work. In contrast, noone really thinks or tries to solve SAT by focusing on algorithms that can be implemented in the restricted circuit models. Thus, along the way we obtain very useful intermediate results that can say powerful things about algorithms for SAT. A disadvantage of this alternative approach is that it may not ever lead to a proof that P is different from NP. This is for two reasons. First, we don't even know if there is an optimal proof system, so it isn't clear how to show that no proof system is polynomially bounded. And secondly, it is not at all clear that NP is different from coNP in the first place.

A second motivation is simple. The complexity of proofs is a fundamental question of logic, and underlies a lot of research in automated theorem proving.

Finally, several other unexpected connections have developed as this area has progressed. For example, there are very interesting links between proof complexity and the complexity of NP search problems. There are also links with learnability and cryptography. We hope to uncover some of these other connections throughout the course.

3 References and Bibliographical notes

- A seminal early paper on propositional proof complexity is by Cook and Reckhow. Cook and Reckhow, The relative efficiency of propositional proof systems. Journal of Symbolic Logic, Vol 44, pp. 36-50, 1979.
- The DPLL procedure was originally proposed by Davis, Putnam, Loveland and Logam. There are hundreds or possibly even thousands of papers on Resolution based proof systems. I will give more references later.
- The Hajos calculus was defined by Hajos in 1961. A more recent paper discussing the complexity of the Hajos calculus is by Pitassi and Urquhart. Pitassi and Urquhart, The complexity of the Hajos Calculus, Siam Journal of Discrete Mathematics, Vol. 8, No. 3, pp.464-483, August 1995.
- The hypergraph transversal problem is a well-known problem in data mining, AI and database circles. It is equivalent to testing equivalence of a monotone CNF and a monotone DNF formula, a well-known problem in theory. A good survey article is: Eiter and Gottlob, Identifying the minimal transversals of a hypergraph and related problems, SIAM Journal on Computing, 24(6), pp. 1278-1304, December 1995. The best known algorithm for the problem (with quasipolynomial runtime) is: Fredman and Khachiyan, On the complexity of dualization of monotone disjunctive normal forms. Technical Report, Department of Computer Science, Rutgers University, 1994.