Survey on Proof Complexity

Adam Chlipala (adamc@cs)

December 17, 2004

1 Introduction

In this survey, I will discuss two results in *proof complexity*. Proof complexity studies the efficiency of particular formal proof systems for proving particular formulas. The work in this area has focused on propositional logic, since results about it have a direct connection to central questions in complexity theory.

In particular, we know that $NP = co - NP$ if there is a proof system that has a small proof for every propositional tautology. If for every propositional contradiction there is a refutation of polynomial size, then we have $\overline{SAT} \in NP$, using these refutations as the NP witnesses. In general, these witnesses could be arbitrary data, and they might not look anything like what we think of as "proofs" in mathematical logic. However, by studying the suitability of particular proof systems for providing these witnesses, we might learn something more fundamental about the problem. Each time we prove that some well-known proof system requires super-polynomial proofs for some contradictions, we accumulate more informal evidence that $NP \neq co - NP$.

The first of the papers I'll be surveying [BSW99] presents a result about the well-known *resolution* proof system. The authors present a useful result that relates a property called *width* to the lengths of resolution proofs. They also describe a general technique for proving proof size lower bounds, based on that result, and show how to use the technique to duplicate or improve a number of past lower bound results. These include a number of exponential lower bounds for propositional tautologies, showing that resolution proofs are not acceptable NP witnesses for $\overline{SAT}$.

The second paper I will discuss [Ats04] builds up to a different kind of result. Instead of fixing a proof system for refuting propositional contradictions, this work considers the case where programs in the *Datalog* language are used to check a formula for unsatisfiability. They consider just how useful a fixed Datalog program can be for this task. The main result establishes that, as the size of input formulas increases, the proportion of formulas that are solutions to any fixed Datalog program that only accepts unsatisfiable formulas tends towards zero. This effectively rules out Datalog programs as well as useful tools for generating NP witnesses for $\overline{SAT}$.

2 Width and resolution

2.1 The resolution proof system

Resolution is one of the simplest known complete proof systems for propositional logic. It has just one proof rule, the *resolution rule*:

$$\frac{A \vee x \quad B \vee \overline{x}}{A \vee B}$$

Here A and B are arbitrary first-order formulas, and x is any propositional variable. This rule isn't purely syntactic. We will be working with formulas in CNF, where we think of a formula as an unordered set of clauses, and of each clause as an unordered set of literals. So, for example, $A \vee x$ can be any clause containing the literal x. We will treat the resolution rule as a way to derive one new clause from two old ones.

We can use this to give a formal definition of a resolution proof of a clause E from hypotheses $\mathcal{F}$: A proof is a sequence $C_1, ..., C_n$ of clauses where $C_n = E$ and, for each C_i, one of the following is true:

- $C_i \in \mathcal{F}$; or
- C_i is derived by the resolution rule from two clauses C_j and C_k, for $j, k < i$.

For instance, say we want to prove x from $(x \vee y \vee z) \wedge (\overline{y} \vee z) \wedge (x \vee \overline{z})$. One proof is:

$$x \vee y \vee z, \overline{y} \vee z, x \vee z, x \vee \overline{z}, x$$

The first, second, and fourth clauses are hypotheses; the third is derived from the first and second by resolution; and the goal is derived from the third and fourth clauses by resolution.

One interesting restricted category of resolution proofs is the *tree-like proofs*. These are roughly the proofs that can be represented with natural deduction-style proof trees. More formally, we can associate with each proof a graph on the clauses it uses, with an edge from each hypothesis of a use of resolution to the corresponding conclusion. A tree-like proof corresponds to graph that is a tree.

The previous example happens to be a tree-like proof, which can be seen by putting it into a natural deduction form:

$$\dfrac{\dfrac{\overline{x \vee y \vee z} \quad \overline{\overline{y} \vee z}}{x \vee z} \quad \overline{x \vee \overline{z}}}{x}$$

However, not all resolution proofs are tree-like. In particular, any that uses the same intermediate clause multiple times as an input to the resolution rule is not tree-like. It is always possible to duplicate parts of the proof to avoid this problem, but this duplication can have a significant effect on proof size. [BSW99] presents a class of formulas that requires exponential size tree-like proofs but admits linear size proofs in general. Tree-like proofs are interesting because many common automated provers, like the Davis-Putnam procedure, generate only tree-like proofs.

Following [BSW99], I'll use the notations $S(\mathcal{F} \vdash C)$ and $S_T(\mathcal{F} \vdash C)$ to denote the textual length of the shortest general or tree-like proof, respectively, of C from $\mathcal{F}$. Since the focus will be on proving that a formula is contradictory, I use $S(\mathcal{F})$ to stand for $S(\mathcal{F} \vdash 0)$ and $S_T(\mathcal{F})$ to stand for $S_T(\mathcal{F} \vdash 0)$. Here, 0 stands for the empty (false) clause.

2.2 Width and its connection to proof size

[BSW99] defines a measure of proof complexity called *width*. The width of a clause C, denoted $w(C)$, is the number of literals in it. The width of a set of hypotheses or of a resolution proof is the maximum width among its clauses. Finally, we can also use width as a measure of the complexity of proving a particular goal. We write $w(\mathcal{F} \vdash C)$ to denote the smallest width of a proof of clause C from hypotheses $\mathcal{F}$.

The main idea of [BSW99] is that often the easiest way to prove that a goal formula has only long proofs is to show that all proofs of it have large width. They prove a formal result that can be used to do this:

Theorem 1. *For any hypotheses $\mathcal{F}$ using n variables, $S(\mathcal{F}) = \exp\left(\Omega\left(\frac{(w(\mathcal{F}\vdash 0)-w(\mathcal{F}))^2}{n}\right)\right)$.*

They also have a tighter result for tree-like proofs:

Theorem 2. *For any hypotheses $\mathcal{F}$, $S_T(\mathcal{F}) \geq 2^{w(\mathcal{F}\vdash 0)-w(\mathcal{F})}$.*

2.3 A strategy for proving width lower bounds

[BSW99] presents the following outline of a method for proving width lower bounds on refutations of some hypotheses $\mathcal{F}$:

1. Define a complexity measure μ mapping clauses to $\mathbb{N}$, such that $\mu(C) \leq 1$ for each $C \in \mathcal{F}$.
2. Prove that $\mu(0)$ is "large".
3. Prove that any refutation of $\mathcal{F}$ must include a clause C with a "medium" μ value.
4. Prove that $w(C)$ is "large" if $\mu(C)$ is "medium".

The particular μ to be used will depend on $\mathcal{F}$, but [BSW99] suggests a parametric definition that is useful for many examples. They define $\mu_{\mathcal{A}}$, where $\mathcal{A}$ is a set of propositional formulas. For any clause C, $\mathcal{A} \models C$ is defined to mean that every variable assignment satisfying every formula of $\mathcal{A}$ also satisfies C.

Definition 1 ($\mu_{\mathcal{A}}$). *For any set $\mathcal{A}$ of formulas and any clause C, $\mu_{\mathcal{A}}(C) \equiv \min\{|\mathcal{A}'| : \mathcal{A}' \subseteq \mathcal{A}, \mathcal{A}' \models C\}$.*

This corresponds with an intuitive notion of hardness. We fix some set of hypotheses $\mathcal{A}$ that is enough to prove our final goal. The complexity of any clause C found in a proof of it is measured by how many of the hypotheses we must assume at a minimum for C to follow. Since we want to use this to show that some propositional contradictions are hard to prove, it should turn out that the goal 0 requires many members of $\mathcal{A}$. If it didn't, it wouldn't be hard to prove.

Lemma 1. *For any clauses B, C, and D, if D can be inferred from B and C with one resolution step, then $\mu_{\mathcal{A}}(D) \leq \mu_{\mathcal{A}}(B) + \mu_{\mathcal{A}}(C)$.*

Proof. Choose $\mathcal{B} \subseteq \mathcal{A}$ such that $|\mathcal{B}| = \mu_{\mathcal{A}}(B)$ and $\mathcal{B} \models B$; and likewise for $\mathcal{C}$ and C. Since adding to a set of hypotheses can only increase the set of consequences, we have $\mathcal{B} \cup \mathcal{C} \models B$ and $\mathcal{B} \cup \mathcal{C} \models C$. Since resolution is a sound inference rule, and D is deduced from B and C by resolution, we have $\mathcal{B} \cup \mathcal{C} \models D$. Thus, $\mu_{\mathcal{A}}(D) \leq |\mathcal{B} \cup \mathcal{C}| \leq |\mathcal{B}| + |\mathcal{C}| = \mu_{\mathcal{A}}(B) + \mu_{\mathcal{A}}(C)$. □

Definition 2 (Compatibility). *For any unsatisfiable set $\mathcal{F}$ of hypotheses, $\mathcal{A}$ is* compatible *with $\mathcal{F}$ if $\mathcal{A} \models 0$ and, for every $C \in \mathcal{F}$, $\mu_{\mathcal{A}}(C) \leq 1$.*

Informally, a particular choice of unsatisfiable $\mathcal{A}$ is compatible with $\mathcal{F}$ if every member of $\mathcal{F}$, which has an "obvious" proof, is also considered trivial under $\mu_{\mathcal{A}}$. Each $C \in \mathcal{F}$ should follow from assuming no more than one member of $\mathcal{A}$.

Lemma 2. *If $\mathcal{A}$ is compatible with $\mathcal{F}$ and $\mu_{\mathcal{A}}(0) \geq 2$, then every proof of $\mathcal{F} \models 0$ contains a clause C with*

$$\frac{1}{3}\mu_{\mathcal{A}}(0) \leq \mu_{\mathcal{A}}(C) \leq \frac{2}{3}\mu_{\mathcal{A}}(0)$$

Proof. Assume for a contradiction that some proof of $\mathcal{F} \models 0$ contains no clause C with $\frac{1}{3}\mu_{\mathcal{A}}(0) \leq \mu_{\mathcal{A}}(C) \leq \frac{2}{3}\mu_{\mathcal{A}}(0)$. Let C be the earliest clause of the proof with $\mu_{\mathcal{A}}(C) > \frac{2}{3}\mu_{\mathcal{A}}(0)$. Such a C exists because 0 itself appears eventually in the proof, and $\mu_{\mathcal{A}}(0) > \frac{2}{3}\mu_{\mathcal{A}}(0)$. Since $\mu_{\mathcal{A}}(0) \geq 2$, $\mu_{\mathcal{A}}(C) > 1$, so $C \notin \mathcal{F}$. Therefore, C must be deduced by resolution from two earlier clauses D and E. By the choice of C, both earlier clauses have width no more than $\frac{2}{3}\mu_{\mathcal{A}}(0)$. By the initial assumption, D and E must have widths below $\frac{1}{3}\mu_{\mathcal{A}}(0)$. By Lemma 1, $\mu_{\mathcal{A}}(C) \leq \mu_{\mathcal{A}}(D) + \mu_{\mathcal{A}}(E)$, a contradiction. □

This takes care of the first three steps in the proof strategy outline. The remaining step uses the idea of an *expansion* to lower bound width. Expansion is based on a series of definitions:

Definition 3 (Sensitivity). *A boolean function f is* sensitive *if, for every variable assignment α with $f(\alpha) = 0$ and any assignment β formed by flipping one variable of α, $f(\beta) = 1$.*

Definition 4 (Critical Assignment). *For any set $\mathcal{A}$ of boolean functions and any $f \in \mathcal{A}$, a* critical assignment *for f is a variable assignment α such that $f(\alpha) = 0$ and $g(\alpha) = 1$ for any $g \in \mathcal{A} \setminus \{f\}$.*

Definition 5 (Dependence). *A boolean function f is* dependent on *a variable x if there are two variable assignments α and β differing only on the value of x such that $f(\alpha) \neq f(\beta)$.*

Definition 6 (Boundary). *Let $\mathcal{A}$ be a set of boolean functions. The* boundary of *$\mathcal{A}$, or $\partial\mathcal{A}$, is the set of variables x such that there is exactly one function in $\mathcal{A}$ dependent on x.*

Now we can define the central idea:

Definition 7 (Expansion). *Let some $\mathcal{A}$ with $\mathcal{A} \models 0$ be given. Define the* expansion of *$\mathcal{A}$ by:*

$$e(\mathcal{A}) = \min\{|\partial\mathcal{A}'| : \mathcal{A}' \subset \mathcal{A}, \frac{1}{3}\mu_{\mathcal{A}}(0) \leq |\mathcal{A}'| \leq \frac{2}{3}\mu_{\mathcal{A}}(0)\}$$

Now we can use expansion to lower bound width. First, one last lemma:

Lemma 3. *If $f \in \mathcal{A}$ is sensitive, α is a critical assignment for f, and $x \in Vars(f) \cap \partial\mathcal{A}$, then flipping the value of x in α yields an assignment β that satisfies $\mathcal{A}$.*

Proof. Let β be the result of flipping the value of x in α. Since α is critical for f, we know that $f(\alpha) = 0$ and $g(\alpha) = 1$ for every $g \in \mathcal{A} \setminus \{f\}$. Because f is sensitive and α and β differ only on x, we have $f(\beta) = 1$. This also shows that f is dependent on x. Since $x \in \partial\mathcal{A}$, we have that no other function in $\mathcal{A}$ is dependent on x. Therefore, since α satisfies all of them, so does β. Thus, β satisfies all of $\mathcal{A}$. □

Theorem 3. *Let $\mathcal{F}$ be an unsatisfiable set of hypotheses. $w(\mathcal{F} \vdash 0) \geq \max e(\mathcal{A})$, where $\mathcal{A}$ ranges over all sets of sensitive functions compatible with $\mathcal{F}$ and with $\mu_{\mathcal{A}}(0) \geq 2$.*

Proof. Let $\mathcal{A}$ as specified in the theorem statement be given, and consider any proof of $\mathcal{F} \vdash 0$. By Lemma 2, there is a clause C in the proof with $\frac{1}{3}\mu_{\mathcal{A}}(0) \leq \mu_{\mathcal{A}}(C) \leq \frac{2}{3}\mu_{\mathcal{A}}(0)$. Let $\mathcal{A}' \subset \mathcal{A}$ be a minimal set such that $\mathcal{A}' \models C$. By definition, $|\mathcal{A}'| = \mu_{\mathcal{A}}(C)$.

Let $x \in \partial\mathcal{A}'$ be given. Consider each $f \in \mathcal{A}'$. There must be an assignment α_f such that $C(\alpha_f) = f(\alpha_f) = 0$ and $g(\alpha_f) = 1$ for all $g \in \mathcal{A}' \setminus \{f\}$. If not, we can derive $\mathcal{A}' \setminus \{f\} \models C$ as follows: This could only be false if there existed an α such that $g(\alpha) = 1$ for all $g \in \mathcal{A}' \setminus \{f\}$ but $C(\alpha) = 0$. We know that $f(\alpha) = 1$, so such a counterexample would also show $\mathcal{A}' \not\models C$. Since this is contradictory, no such counterexample can exist. Thus, we could have chosen $\mathcal{A}' \setminus \{f\}$ instead of $\mathcal{A}'$, which contradicts $\mathcal{A}'$'s minimality, so an α_f as posited must exist.

Now suppose for a contradiction that $x \in Vars(f) \cap \partial\mathcal{A}'$ but x does not appear in C. Let β be the result of flipping the value of x in α_f. We have that f is sensitive (because it's in $\mathcal{A}$), α_f is a critical assignment for f, and $x \in Vars(f) \cap \partial\mathcal{A}'$. Therefore, by Lemma 3, β satisfies $\mathcal{A}'$. Since x does not appear in C, β still does not satisfy C. Therefore, $\mathcal{A}' \not\models C$, a contradiction.

This shows that every variable in $\partial\mathcal{A}'$ also appears in C. Thus, $w(C) \geq |\partial A'|$. By the definition of boundaries, $|\partial A'| \geq e(\mathcal{A})$, since $\frac{1}{3}\mu_{\mathcal{A}}(0) \leq |\mathcal{A}'| \leq \frac{2}{3}\mu_{\mathcal{A}}(0)$. Therefore, $w(C) \geq e(\mathcal{A})$.

Since this argument works for every proof of $\mathcal{F} \vdash 0$, we have $w(\mathcal{F} \vdash 0) \geq e(\mathcal{A})$. Since the argument works for every valid $\mathcal{A}$, we have the final result. □

2.4 Example: A lower bound for Tseitin Formulas

The simplest application of this technique that [BSW99] presents involves *Tseitin Formulas*:

Definition 8 (Tseitin Formulas). *Let G be a finite connected graph. A function $f : V(G) \to \{0,1\}$ has* odd weight *if $\sum_{v \in V(G)} f(v) \equiv 1 \pmod 2$. Let $d_G(v)$ be the degree of vertex v in G. Fix an odd weight function f. Associate a variable x_e with each edge of G. For $v \in V(G)$, define $PARITY_v \equiv (\bigoplus_{v \in e} x_e \equiv f(v) \pmod 2)$. The* Tseitin Contradiction *of G and f is:*

$$\tau(G, f) = \bigwedge_{v \in V(G)} PARITY_v$$

Each $PARITY$ must be represented with a number of clauses. [BSW99] cites without proof a lemma that characterizes just how efficiently $\tau(G, f)$ can be represented in CNF:

Lemma 4. *If d is the maximal degree of G, then $\tau(G, f)$ is a d-CNF with at most $n2^{d-1}$ clauses and $\frac{nd}{2}$ variables.*

A definition of another type of expansion is introduced, this time over graphs:

Definition 9 (Expansion). *For $G = (V, E)$ a finite connected graph, the* expansion *of G is:*

$$e(G) = \min\{|E(V', V \setminus V')| : V' \subseteq V, \frac{1}{3}|V| \leq |V'| \leq \frac{2}{3}|V|\}$$

Here, $E(V', V \setminus V')$ denotes the cut between V' and $V \setminus V'$ in G. Graph expansion characterizes how effectively a graph can be partitioned into two mostly disjoint pieces of roughly equal size.

The main theorem about expansion and weight uses this lemma from a previous paper:

Lemma 5. *If G is connected, then $\tau(G, f)$ is contradictory iff f is an odd weight function.*

Theorem 4. *For G a connected graph and f an odd-weight function on $V(G)$, $w(\tau(G, f) \vdash 0) \geq e(G)$.*

Proof. They define $\mathcal{A}_V = \{PARITY_v : v \in V(G)\}$ and $\mu = \mu_{\mathcal{A}_V}$, and proceed to apply Theorem 3. We can check each requirement of that theorem:

$\mathcal{A}_V$ contains only sensitive functions. Each member of $\mathcal{A}_V$ is a parity function, which is clearly sensitive: changing any input to a parity function changes its output.

$\mathcal{A}_V$ is contradictory. This is trivial, since the conjunction of $\mathcal{A}_V$'s elements is precisely a Tseitin Contradiction.

For every $C \in \tau(G, f)$, $\mu(C) \leq 1$. Each C is one of the clauses used to encode $PARITY_v$ for some $v \in V(G)$. Since $PARITY_v \in \mathcal{A}_V$ and $\{PARITY_v\} \models C$, we have $\mu(C) = 1$.

$\mu(0) \geq 2$. This is trivial if $|V(G)| \leq 2$. Otherwise, assume for a contradiction that there is $\mathcal{A} \subseteq \mathcal{A}_V$ with $|\mathcal{A}| = 1$ such that $\mathcal{A} \models 0$. $\mathcal{A}$ must be $\{PARITY_u\}$ for some $u \in V(G)$. Define $V' = V(G) \setminus \{u\}$. Choose $v \in V'$. Define $f'(v)$ to be $1 - f(v)$ and $f'(w)$ to be $f(w)$ when $w \neq v$. Since f has odd weight, f' does not. Therefore, by Lemma 5, $\tau(G, f')$ is satisfiable. $\mathcal{A}_{\{u\}}$ is a sub-formula of $\tau(G, f')$, so it is satisfiable as well. Thus, every size-1 subset of $\mathcal{A}_V$ is satisfiable, so $\mu(0) \geq 2$.

So, by Theorem 3, $w(\tau(G, f) \vdash 0) \geq e(\mathcal{A}_V)$. Any subset $\mathcal{A}' \subseteq \mathcal{A}_V$ is $\mathcal{A}_{V'}$ for some $V' \subseteq V$. Consider such a V'. $\partial\mathcal{A}_{V'} = \{x_e : e \in E(V', V \setminus V'\}$, because clearly any $PARITY_v$ is dependent on the variable for any edge incident on it; and any other edge variable appearing in $\mathcal{A}_{V'}$ besides those listed must connect two elements of V', meaning that multiple elements of $\mathcal{A}_{V'}$ depend on it. Thus, by the definitions of the two kinds of expansion, $e(\mathcal{A}_V) \geq e(G)$, so $w(\tau(G, f) \vdash 0) \geq e(G)$. □

From this result, [BSW99] re-proves an old result which says: If G is 3-regular and $e(G) = \Omega(|V(G)|)$, then $S(\tau(G, f)) = 2^{\Omega(|\tau(G,f)|)}$. To derive this, we can first use Theorem 1 to deduce $S(\tau(G, f) \vdash 0) = exp\Big(\Omega\Big(\frac{(e(G)-w(\tau(G,f)))^2}{|E(G)|}\Big)\Big) = exp\Big(\Omega\Big(\frac{(|V(G)|-3)^2}{3|V(G)|/2}\Big)\Big) = exp(\Omega(|V(G)|))$.

3 Refuting random instances of 3SAT

Resolution proofs are one way of encoding witnesses for the unsatisfiability of formulas. To determine whether or not $\overline{3SAT} \in NP$, we need somehow to consider *all* possible proof systems, so it is natural to study successively more complex mechanisms to see what we can learn about the general case. [Ats04] considers what can be accomplished using programs in the *Datalog* language to encode procedures for determining formula unsatisfiability. Datalog is not a Turing-complete language, but it is expressive enough for a large variety of procedures, including resolution.

The high-level result of [Ats04] is that no Datalog program can be very useful for detecting unsatisfiability in random 3CNF formulas. Any Datalog program that only accepts unsatisfiable inputs will tend asymptotically towards only accepting a negligible fraction of allowed inputs.

3.1 Datalog

The Datalog language is a subset of the well-known logic programming language Prolog. While Prolog allows rules to build new expressions by applying function symbols, in Datalog variables are the only expressions. It is easy to compensate for this restriction by using relations to stand for particular functions. For instance, the Prolog rule

$$length(cons(x, y)) = 1 + length(y).$$

expresses the recursive case of a length function for lists. We can use relations instead of the functions $length$ and $cons$ to obtain a Datalog equivalent:

$$length(l, z) \dashv cons(x, y, l), length(y, l'), add(1, l', z).$$

Datalog equivalents of Prolog programs can require significantly more variables, which will be important later. However, the expressiveness is still quite good. [BSW99] mentions unsatisfiability of 2CNF instances, graph reachability, and non-2-colorability as some computational problems solvable by Datalog programs.

The results of [Ats04] use number of distinct variables in Datalog programs as the crucial computational resource. They define a *k-Datalog program* to be one using at most k distinct variables and using no relation symbol with arity greater than k.

3.2 Characterizing Datalog programs with infinitary logic

To simplify the overall argument, [Ats04] reduces Datalog programs to formulas in a particular infinitary logic, called $\exists L^k_{\infty\omega}$. Atomic formulas of $\exists L^k_{\infty\omega}$are applications of constant relation symbols or second-order variables (which stand for unknown relations) to first-order variables (which stand for unknown individuals in some underlying universe). The full set of formulas is the closure of this set of atomic formulas under conjunction (possibly with infinitely many conjuncts), (possibly infinite) disjunction, and existential quantification over first-order variables. Because negation and universal quantification are left out, this logic is called *positive existential* infinitary logic. The "k" in the name denotes a limit of k distinct variables in formulas.

[Ats04] cites a result that says that, for any k-Datalog program P, there is a formula φ of $\exists L^k_{\infty\omega}$such that the set of solutions to P is precisely the set of satisfying assignments for φ. Thus, the rest of the development focuses on stronger results about the inability of $\exists L^k_{\infty\omega}$formulas to express useful tests for unsatisfiability.

3.3 Encoding 3SAT in first-order logic

We want to use first-logic to reason about propositional formulas. Since we are only interested in using first-order logic to express this reasoning, we will need to encode propositional formulas in some way, instead of reasoning about them directly. The strategy [Ats04] takes is to identify 3SAT formulas with particular models for a fixed logical vocabulary.

We fix a language of 3-ary relation symbols R_0, R_1, R_2, and R_3. Every input 3CNF F will correspond with a particular *model* $M(F)$ for this language. $M(F)$'s universe is the set of variables appearing in F. R_i will stand for those clauses of the input with exactly i negated literals. Whenever the input contains a clause:

$$\neg x_1 \vee \ldots \vee \neg x_i \vee y_1 \vee \ldots \vee y_{3-i}$$

we say that:

$$R_i(x_1, \ldots, x_i, y_1, \ldots, y_{3-i})$$

R_i does not hold for any other arguments. Since we consider clauses to be unordered, a given clause will in this way contribute multiple true argument lists to a particular R_i.

[Ats04] also defines another model for the same language, a *template structure* called T. While the previous kind of models stand for particular inputs, T will represent the properties of satisfiable 3CNF's. We define the universe of T to be the truth

assignments $\{0, 1\}$. Each R_i is again associated with clauses of exactly i negated variables, but now we must define their behavior over inputs that are truth values. We will have each R_i interpret its inputs as truth assignments to the variables that appear in some clause with i negated variables. The first i inputs give the truth values of the negated variables, and the next $3 - i$ give truth values to the non-negated variables. Thus, R_0 accepts every input but $(0, 0, 0)$, R_1 every input but $(1, 0, 0)$, and so on.

With these definitions, we can express the satisfiability of a 3CNF F in terms of a relationship between that formula's model and the template T. F is satisfiable iff there is a *homomorphism* from $M(F)$ to T. A homomorphism is a function from the universe of one model to the universe of another that respects the two models' interpretations of the relation symbols. In this case, this would mean a function f from $Vars(F)$ to $\{0, 1\}$ such that, if $R(x_1, x_2, x_3)$ in $M(F)$, then $R(f(x_1), f(x_2), f(x_3))$ in T. Such a homomorphism is precisely a satisfying assignment of F.

3.4 A connection with a combinatorial game

[Ats04] presents a combinatorial game, called the *existential k-pebble game*, that provides a different way of thinking about homomorphisms. The game has two players. One player, called the Duplicator, is trying step-by-step to construct a homomorphism between two structures. His adversary, the Spoiler, is trying to prevent him from accomplishing this.

Each game is played over two logical structures A and B for the same language. Each player has k numbered pebbles. The game is played on a "board" made up of the elements of the universes of A and B. For each round of the game, Spoiler can either place one of his unused pebbles on an element of A that has no pebble on it, or Spoiler can remove one of his pebbles from an element of A. Duplicator's job is to mimic Spoiler's action. If Spoiler placed pebble i on an element of A, Duplicator must place his pebble numbered i on some unpebbled element of B. If Spoiler removed pebble i, Duplicator must remove his pebble i as well.

Duplicator's goal is to maintain the invariant that, after each round, the configuration of pebbles denotes a *partial homomorphism* from A to B. For any configuration, we can define a function f mapping the pebbled elements of A to the pebbled elements of B, where the element with Spoiler's pebble i is mapped to the element with Duplicator's pebble i. This mapping is a partial homomorphism if Duplicator could place the remaining pebbles however he likes, without moving the already-placed pebbles, such that the resulting f is a homomorphism.

If Duplicator is ever forced to create a configuration that does not denote a partial homomorphism, Spoiler wins the game. Duplicator wins when he is able to continue the game forever.

[Ats04] cites a result that says, informally, that this game has exactly the power of $\exists L^k_{\infty\omega}$to distinguish between structures.

Theorem 5. *Let L be a relational vocabulary, $\mathcal{C}$ a class of finite structures for L, and Q a Boolean query on $\mathcal{C}$. The following are equivalent:*

1. *Q is definable in $\exists L^k_{\infty\omega}$on $\mathcal{C}$.*

2. *For every two structures $A, B \in \mathcal{C}$, if $A \models Q$ and Duplicator wins the existential k-pebble game on A and B, then $B \models Q$.*

So now, by way of the last section, we have a connection between expressibility in Datalog and the winnability of this game.

3.5 Another game

[Ats04] now develops some results about a similar game which had been studied in previous work, with the goal of importing these results. The new game, called the *matching game on G with k fingers*, deals with a bipartite graph $G = (U \cup V, E)$. The structure of the game is very similar to the existential k-pebble game. Two players Prover and Disprover take the roles of Spoiler and Duplicator, respectively. Each has k numbered pebbles, and the board is composed of the vertices of G. On each round, Prover places a pebble over an element of U or removes such a pebble. Disprover must respond by placing his same-numbered pebble on an unpebbled neighbor of U or removing his same-numbered pebble from the board, respectively. Prover wins if Disprover is ever unable to make a legal move, and Disprover wins if he can prolong the game forever.

For given $s \in \mathbb{N}$ and $\epsilon > 0$, we say that G is an (s, ϵ)-*bipartite expander* if every $A \subseteq U$ of size at most s has at least $(1 + \epsilon)|A|$ neighbors in V. We also define the *left-degree* of G to be the largest degree of any member of U, with an analogous definition for right-degree and V. The main result about this game used by [Ats04] is:

Theorem 6. *Let G be an (s, ϵ)-bipartite expander of left-degree at most l, and let r be a positive integer bounded by $\frac{\epsilon s}{l+\epsilon}$. Disprover wins the matching game on G with r fingers.*

3.6 Putting it together

To be able to make use of this result, [Ats04] defines a bipartite graph $G(F)$ for every 3CNF F. One side of the graph has nodes corresponding to the clauses of F, and the other side is the variables of F. We place an edge from each clause to each variable appearing in it. Now we have the result:

Theorem 7. *Let F be a $3CNF$ such that $G(F)$ has right-degree at most d. If Disprover wins the matching game with r fingers on $G(F)$, then Duplicator wins the existential $\lfloor \frac{r}{d} \rfloor$-pebble game on $M(F)$ and T.*

This is the last tool needed to prove the main theorem of the paper:

Theorem 8. *For all constants $\delta > 0$ and $\Delta > 0$, and for all sufficiently large n, if C_n is a set of 3CNF's with no more than n variables, and*

1. *Every member of C_n is unsatisfiable.*
2. *$\Pr[M(F) \in C_n] \geq \delta$ when F is drawn uniformly from the set of possible 3CNF's with n variables and Δn clauses.*

then C_n is not definable in $\exists L^k_{\infty\omega}$over 3CNF's with n variables for $k \leq \frac{n}{\ln(n)^2}$.

The proof assumes for a contradiction that the theorem is false, so, for some large enough n, we have C_n satisfying both conditions above and also definable in $\exists L^k_{\infty\omega}$with the parameters given. They use the probabilistic method to show that there must exist 3CNF F with $M(F) \in C_n$ and $G(F)$ an $\left(s = \frac{n}{\ln(n)}, \epsilon = \frac{1}{4}\right)$-bipartite expander of right-degree at most $d = \frac{\ln(n)}{13}$ and left-degree at most 3. They use a union bound to do this, proving separate bounds on the probabilities that the right degree of $G(F)$ is bigger than d, that $G(F)$ is not an (s, ϵ)-bipartite expander, and that $M(F) \notin C_n$, this last part provided by assumption 2 of the theorem statement. The sum of these probabilities is proved to be below 1 for the parameters chosen and n large enough, so such an F must exist.

Now, by Theorem 6, Disprover wins the matching game on $G(F)$; and by Theorem 7, Duplicator wins the existential k-pebble game on $M(F)$ and T. This means that, by Theorem 5, T satisfies the $\exists L^k_{\infty\omega}$version of C_n. Since T encodes satisfiability of 3CNF's, we reach the contradictory result that C_n contains a satisfiable formula.

With this result, it is easy to see that no Datalog program can be very useful for finding unsatisfiable formulas. Any Datalog program has only a fixed number k of variables. Theorem 8 shows that, for large enough n, any formula of $\exists L^k_{\infty\omega}$either allows satisfiable formulas or doesn't find many formulas. Since any k-Datalog program is representable in $\exists L^k_{\infty\omega}$, we have that such a program is not very useful past a certain input size.

References

[Ats04] Albert Atserias. On sufficient conditions for unsatisfiability of random formulas. *J. ACM*, 51(2):281–311, 2004.

[BSW99] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow: resolution made simple. In *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 517–526. ACM Press, 1999.