SIGACT News Complexity Theory Column 34

Lane A. Hemaspaandra
Dept. of Computer Science, University of Rochester
Rochester, NY 14627, USA lane@cs.rochester.edu

Introduction to Complexity Theory Column 34

This issue's guest columnists are angels. The column was faced with a last-minute cancellation, and they prepared on very short notice a wonderful column. And I'm sure that many readers of the column will want to learn even more by, for example, reading their monograph *Complexity Classifications of Boolean Constraint Satisfaction Problems* [7].

Future complexity columns include Holzer/McKenzie on a familiar complexity class turning up in a surprising context, Gasarch (and a host of contributors) on the future of NP, Schaeffer/Umans on completeness for higher levels of the polynomial hierarchy, Nickelsen/Tantau on partial information classes, and Paturi on the complexity of k-SAT,

Regarding the Schaeffer/Umans column, their 2-part column will be in part a kind of Garey and Johnson for completeness results for Σ_i^p and Π_i^p, $i \geq 2$. You can help their project by, if you have proven natural sets complete for $\mathrm{NP}^{\mathrm{NP}}$, $\mathrm{coNP}^{\mathrm{NP}}$, $\mathrm{NP}^{\mathrm{NP}^{\mathrm{NP}}}$, $\mathrm{coNP}^{\mathrm{NP}^{\mathrm{NP}}}$, etc., letting them (`MSchaefer@cti.depaul.edu` and `umans@microsoft.com`) know of your result and its best citation location. Thanks!

Guest Column: Complexity Classifications of Boolean Constraint Satisfaction Problems[1]

Nadia Creignou[2] *Sanjeev Khanna*[3] *Madhu Sudan*[4]

Abstract

Many fundamental combinatorial problems can be formulated as Boolean constraint satisfaction problems. Roughly speaking, an instance of such a problem consists of a collection of simple constraints on Boolean variables. Some typical computational goals include finding an assignment that satisfies all constraints, counting the number of satisfying assignments, or finding an "optimal" satisfying assignment. This article surveys some results that give a complete taxonomy of complexity of computational problems that arise in this framework. These results highlight the fact that Boolean constraint satisfaction problems offer an excellent framework to study central problems in a host of standard complexity classes and provide formal basis

[2]Laboratoire d'Informatique Fondamentale, Université de la Méditerranée, Marseille, 13288, FRANCE. E-mail: creignou@lim.univ-mrs.fr.

[3]Dept. of Computer & Information Science, University of Pennsylvania, Philadelphia, PA 19104, USA. E-mail: sanjeev@cis.upenn.edu. Supported in part by an Alfred P. Sloan Research Fellowship and by an NSF Career Award CCR-0093117.

[4]Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. E-mail: madhu@mit.edu. Supported in part by NSF grants CCR-9875511, CCR-9912342 and by MIT-NTT award MIT2001-01.

for "empirical observations" concerning computational phenomena in these classes. We also describe a unifying toolkit for obtaining such taxonomy results for a variety of computational goals.

1 Introduction

One of the fundamental goals of complexity theory is to classify computational problems as easy or hard. Other goals include determining notions of "easiness" or "hardness"; proving that different notions of hardness may or may not coincide (such as, say, showing NP different from P); and to see what makes problems hard or easy. At the moment we have plenty of notions of hardness and few clues on how to prove that some of these notions may or may not coincide. Let us suppose that some day we realize our dreams of proving as strong lower bounds as we could hope for, for specific problems. Even in this wildly optimistic scenario, we do not expect complexity theory to culminate with a compact classification describing which problems are easy and which ones hard. We will continue to need to investigate individual problems to ascertain their computational complexity.

There are several reasons why a complete taxonomy of computational problems is impossible. To start with, there are infinitely many computational problems. Furthermore, Rice's theorem says that given a completely general computational problem (by say a Turing machine deciding it), determining the minimal computational complexity required to solve the problem is undecidable. Even worse, the computational complexity of the problem may not have one of finitely many possibilities. The famed theorem of Ladner [17] shows, for instance, that if NP$\neq$P, then there are infinitely many incomparable problems in NP. (Specifically, the equivalence relationship induced by polynomial time interreducibility splits into infinitely many equivalence classes.) So there is no hope for a taxonomy of computational problems!

One may give up completely upon hearing this terrible news; or like some brave researchers (authors of this article included) try to find a way around this obstacle. The big barrier in finding complete classifications of the computational complexity of problems is the generality of the computational problems considered. These problems are so general as to include the classification task itself, and thus allow diagonalization and then follows a host of impossibility results. Thus, to make some positive steps in the direction of classification, one has to limit one's scope and restrict the class of problems considered to a relatively simple (but still infinitely large) class. Results of such a nature—that consider restricted subclass of problems in well-studied complexity classes, and give simple rules that can be applied to extract the computational complexity of every problem in the class—are what we call *complexity classifications*. In this article we will survey some of these classifications (and ahem ... plug for a research monograph [7] written by the authors on this topic).

1.1 Brief History

The first complexity classification result is a beautiful result due to Schaefer [20]. Schaefer generalizes the classical 3-SAT problem naturally to allow other forms of simple Boolean constraints on Boolean variables. For each class of constraints allowed, he derives a different computational problem, whose complexity he examines. Schaefer's paper is a well-referenced one in the theory literature. Most often it is cited for showing that "not-all-equal-three-sat" or "one-in-three-sat" is NP-complete. These are just minor corollaries to his main result which pins down the complexity of every one of the problems considered. On purely technical grounds, this result is non-trivial. A reader wishing to verify this, may try proving the NP-hardness of "one-in-three-sat" on his/her own and seeing the level of complexity that enters! But the paper actually tackles much harder issues.

For example, the exact nature of the "constraints" used in specifying a constraint satisfaction problem seem to be a feature of an *instance*, rather than of a *computational problem*, and yet as we well know, complexity is a characteristic of the problem and not an instance. Dealing with such issues requires significant care, and Schaefer deals with them naturally. In the process, Schaefer derives, completely naturally, a subclass of NP that exhibits an amazing *dichotomy*—every problem in this class is either in P, or is NP-complete.

There seems to have been no follow-up work on Schaefer's through the eighties, but the nineties saw a diverse collection of researchers motivated by varying interests following up on Schaefer's work. Dichotomy or, more generally, classification results are of interest for a multiple reasons. On the one hand they give a compact presentation of known results. So it is reasonable to ask what classes (other than NP) contain classes that exhibit dichotomy. This issue motivated the work of Creignou et al. [4, 5, 6] and Hunt et al. [9]. Within NP, one can wonder if larger classes exhibit dichotomy—this issue was considered by Feder and Vardi [8] and Jeavons et al. [11]. Further, the reductions used in these classification results become interesting in their own, and this motivated the study of Hunt et al. [10], as also the monograph by the authors [7]. In fact, a tangential result along these lines shows how such reductions may even be found by an automated search (see Trevisan et al. [23]). Khanna et al. [15] show how to use classification results to unify and extend many results in the study of the approximability of optimization problems. We will discuss these results in some detail shortly. First we introduce constraint satisfaction problems.

2 Constraint Satisfaction Problems

We introduce constraint satisfaction problems informally by recalling the prototypical example, namely 3-SAT. An instance ϕ of 3-SAT consists of m clauses $C_1, \ldots, C_m$ on n Boolean variables $x_1, \ldots, x_n$. Each clause is a disjunction of up to 3 literals, where a literal is just a variable, or its negation. An assignment to the n variables to either "true" or "false" induces an assignment to the literals naturally. Such an assignment satisfies a clause C_j if at least one of the literals contained in the clause is set to true. It satisfies the formula if all clauses are satisfied. Thus each clause becomes a constraint on the assignment and the computational problem is that of deciding if there exists an assignment that satisfies all constraints.

Schaefer considered a generalized family of satisfiability problems, where problems differed in the nature of constraints that were allowed. Care has to be taken to separate out the *template* of the constraints allowed, from their *applications* to some ordered subset of all variables. The basic templates that are allowed specify the *satisfiability problem*, while a specific instance consists of a collection of applications of such constraints. For example, by restricting all clauses to have length at most 2—a restriction that changes the *problem*—we get the 2-SAT problem, which is significantly easier. Other possible constraint satisfaction problems could include the problem where every constraint is a linear constraint over GF(2), or where every constraint "monotone" is a clause involving only positive literals. Both variants give constraint satisfaction problems in P. However if we allow a mix of linear constraints and monotone constraints then we get an NP-hard problem! Schaefer studied all such constraint satisfaction problems, which he called "generalized satisfiability problems" collectively. He gave simple rules showing which ones were NP-hard and which ones were in P, and further showed there were no intermediate problems.

To formally define the class of problems that Schaefer considered, we have to formalize the notion of constraints and applications. We do so below. Here onwards we represent Boolean values by 0 and 1, with 0 representing the logical false.

Definition 2.1 (Constraints and Constraint Families) *A constraint is just a function $f : \{0,1\}^k \to \{0,1\}$ given by a truth table, for some finite k. The* arity *of such a constraint is k. A constraint family $\mathcal{F}$ is a finite collection of constraints.*

Definition 2.2 (Constraint applications) *Given n Boolean variables $x_1, \ldots, x_n$ and a constraint $f : \{0,1\}^k \to \{0,1\}$, a function $C : \{0,1\}^n \to \{0,1\}$ is an* application *of f to $x_1, \ldots, x_n$ if there exists indices $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $C(x_1, \ldots, x_n) = f(x_{i_1}, \ldots, x_{i_k})$.* [5]

An application C is satisfied by an assignment to the variables if the constraint evaluates to 1 on the assignment. We are now ready to define the generalized satisfiability problems of Schaefer [20].

Definition 2.3 *For a constraint family $\mathcal{F}$, the generalized satisfiability problem, $\mathsf{SAT}(\mathcal{F})$, is the problem whose instances consist of m applications of constraints from $\mathcal{F}$ to n Boolean variables $x_1, \ldots, x_n$, and the goal is to decide if there exists an assignment to $x_1, \ldots, x_n$ satisfying all constraints.*

As an example, let us consider the family that leads to the 3-SAT problem: For $j \leq i$, let $\mathsf{OR}_{i,j}$ denote the function $\mathsf{OR}_{i,j}(x_1, \ldots, x_i) = x_1 \vee x_2 \vee \cdots \vee x_j \vee \neg x_{j+1} \vee \cdots \vee x_i$. Let $\mathcal{F}_{\mathsf{3\text{-}SAT}} = \{\mathsf{OR}_{3,0}, \mathsf{OR}_{3,1}, \mathsf{OR}_{3,2}, \mathsf{OR}_{3,3}\}$. Then 3-SAT is the problem $\mathsf{SAT}(\mathcal{F}_{\mathsf{3\text{-}SAT}})$. Note that negations do not come for free in this world—so it takes four different functions to cover all different types of clauses.

We now start to describe Schaefer's dichotomy theorem. To motivate this theorem, recall that there are examples of families $\mathcal{F}$ for which $\mathsf{SAT}(\mathcal{F})$ is in P, while others for which $\mathsf{SAT}(\mathcal{F})$ is NP-hard. Below, we list a collection of different reasons why a $\mathsf{SAT}(\mathcal{F})$ problem may be easy. As should be expected, these reasons identify something special in the family $\mathcal{F}$.

- A constraint f is 0-*valid* if $f(0, \ldots, 0) = 1$. A family $\mathcal{F}$ is 0-valid if all functions in $\mathcal{F}$ are 0-valid. Similarly f is 1-*valid* if $f(1, \ldots, 1) = 1$ and $\mathcal{F}$ is 1-valid if all functions in $\mathcal{F}$ are 1-valid.

- A constraint f is *bijunctive* if it can be expressed as a 2-CNF formula. A family is bijunctive if all its members are bijunctive.

- A constraint is *affine* if it can be expressed as a conjunction of linear constraints over GF(2). A family is affine if all its members are affine.

- A constraint is *weakly positive* if it can be expressed as a CNF formula with at most one negated literal in each clause. A family is weakly positive it all its elements are weakly positive. Similarly, a constraint is *weakly negative* if it can be expressed as a CNF formula with at most one unnegated literal in each clause. A family is weakly negative it all its elements are weakly negative.

Schaefer notes that if $\mathcal{F}$ is 0-valid or 1-valid or bijunctive or affine or weakly positive or weakly negative then $\mathsf{SAT}(\mathcal{F})$ can be decided in polynomial time. The amazing theorem of Schaefer, is that for any family that does not fall in one of the above categories, the $\mathsf{SAT}(\mathcal{F})$ problem is hard!

[5] Khanna et al. [15] take a somewhat more stringent view on these indices and insist that $i_1, \ldots, i_k$ should be distinct. This gives them a more refined collection of problems with ability to separate e.g., the exact 3-sat problem from the 3-sat problem. Here we will allow i_j's to be repeated.

Theorem 2.4 ([20]) *For every constraint family $\mathcal{F}$, the generalized satisfiability problem* $\mathsf{SAT}(\mathcal{F})$ *is either in* P *or is* NP*-complete. Furthermore* $\mathsf{SAT}(\mathcal{F})$ *is in* P *if* $\mathcal{F}$ *is* 0*-valid or* 1*-valid or bijunctive or affine or weakly positive or weakly negative, and is* NP*-complete otherwise.*

We will describe some of the essence of the proof later; we first describe some of the subsequent works in detail.

3 Extending and Generalizing Schaefer's Results

Schaefer's result inspired many researchers to study various other complexity classes restricted to constraint satisfaction problems. Here we list some of the directions of work, and the underlying motivations.

- Satisfiability, in some appropriate variant, is the commonest example of a complete problem for many classes including PSPACE, #P, etc. So it is natural to expect that these classes will also contain a subclass of problems that are variants of generalized satisfiability problems and that these problems will exhibit a dichotomy. This direction has been investigated in a number of different works including those of Hunt, Marathe and Stearns [9], Creignou [4], Kavvadias and Sideri [12], Creignou and Hermann [6], Creignou and Hébrard [5], Reith and Vollmer [19], Böhler, Hemaspaandra, Reith, and Vollmer [2].

  Together, these results give dichotomy theorems for subclasses of a wide variety of complexity classes including #P, PSPACE, MAX SAT etc. In almost all cases, the easy problems in the dichotomy are not the same as the easy cases of Schaefer's problems—making these results quite non-trivial.

- Within NP, can we find larger classes of problems can exhibit dichotomy? Jeavons et al. [11] examine this issue by expanding the scope of satisfiability problems to the case of *non-Boolean* variables. Note that as the domain of the variables becomes larger, the scope of constraint satisfaction problems gets richer, and so classification theorems are harder (the more the problems, the harder the classification). Jeavons et al. do not obtain classification theorems, and the issue of whether a dichotomy occurs even for constraint satisfaction problems on ternary variables, remains open to this date.

- Feder and Vardi [8] take an opposite path to the search for the "largest" subclass of NP that might exhibit dichotomy. They approach the issue from the syntactic perspective. Starting with Fagin's syntactic characterization of NP, they work their way down the syntax, ruling out various operators in the syntactic language and converge upon a syntactic subclass of NP—termed "monotone monadic strict NP without inequality"—that *might* exhibit dichotomy. They show that without all their restrictions, the ensuing class is too general for a dichotomy to apply (in particular, Ladner's theorem remains applicable). With all the restrictions, they do not obtain a dichotomy, but they do show every problem in their class is equivalent (under randomized polynomial time reductions) to constraint satisfaction problems on some finite (but non-Boolean domains), thereby converging to the problems studied by Jeavons et al.

- One way to interpret constraint satisfaction problems, and dichotomy theorems is that these give a bird's-eye view of central phenomena within complexity classes. This perspective is especially useful in the study of the approximability of NP-hard optimization problems, where approximability comes in too many flavors and it is hard to summarize salient phenomena. Khanna et al. [15], following up on Creignou's work [4], show how to use variants of Boolean constraint satisfaction problems to summarize a wide variety of recent approximability/inapproximability results, while also obtaining complete classification results. These results are not dichotomies any more, since optimization problems admit more than two possibilities in terms of their approximability. However, the ensuing classifications manage to classify the approximability of problems into one of a finite number of distinct flavors. (Making this precise requires work and the reader is pointed to the actual paper [15] to get more details.)

- Another interesting feature that is highlighted nicely by this "bird's eyeview" of complexity is the nature of reductions that we see "most often". Often the same "reduction", say between 3-SAT and "not-all-equal-3-sat", can be used to show NP-hardness, #P-hardness, or PSPACE-hardness etc. of appropriate variants of the not-all-equal-3-sat problem. This phenomenon is pervasive, and applies to reductions used between many pairs of problems. In the world of constraint satisfaction problems, this phenomenon can be articulated and explained formally. This has been done in several ways in the literature. For instance, in the monograph by the authors [7], the central theme is abstracted in the form of implementations or gadget reductions (following [15]) and then used to prove all classification results in a unified way. In fact, once the scope is identified, it is even possible to "search" for such a reduction using a computer, as shown by Trevisan et al. [23]. In a different direction, Hunt et al. [9, 21, 10], show how the existence of reductions with a collection of locality properties, simultaneously implies reductions between corresponding pairs of all variants of two given problems.

As the reader may realize, not all the work above has led to classification theorems, and in some cases it is not even the target of the research. The research that has led to dichotomies or classifications, is the focus of a recent monograph by the authors [7]. The primary focus of this monograph is to extract the common themes behind the proof of the many classification results. Below we summarize some of the common themes and give an example of how a classification theorem may be obtained.

4 Basic Tools

One of the surprising aspects of dealing with constraint satisfaction problems is that even though there exist so many variations of the main theme; the tools used for dealing with them are essentially the same. In order to establish a complexity classification theorem, our goal is to obtain results of the form: "If all constraints in a given constraint family $\mathcal{F}$ satisfy a certain property $\mathcal{P}$, then a canonical algorithm can be applied to solve the problem efficiently. Otherwise, the constraints in $\mathcal{F}$ that do not satisfy the property can be used to show that the constraint satisfaction problem associated with $\mathcal{F}$ is hard". Typically, most of the effort is involved in establishing the latter part, that is, the hardness result. We briefly describe here a basic framework for establishing the hardness results.

The central mechanism for establishing hardness of a problem is via reductions. Since problems in our framework are specified via underlying constraint sets, the notion of a reduction between problems essentially corresponds to transformation between constraint sets. Towards this end, a natural notion is the idea of *implementation* which is a mechanism for showing that the behavior of a given constraint can be "implemented" by some collection of functions. This notion was explicitly formalized in [13] and its variants have been implicitly used by other researchers (see [1], [6], and [23], for instance). The precise definition of implementation depends on the specific class of constraint satisfaction problems that we are examining. As a concrete example, let us consider the class $\mathsf{SAT}(\mathcal{F})$. We say that a constraint family $\mathcal{F}$ implements a function $f(\vec{x})$ if there exists an instance of $\mathsf{SAT}(\mathcal{F})$ defined over $\vec{x}$ and possibly an auxiliary set of variables $\vec{y}$ such that (a) for any assignment $\vec{x}$ such that $f(\vec{x})$ is true, there exists an assignment to $\vec{y}$ that satisfies all constraints, and (b) for any assignment $\vec{x}$ such that $f(\vec{x})$ is false, no assignment to $\vec{y}$ can satisfy all constraints. For instance, the family $\{\mathsf{XOR}\}$ implements the function $\mathsf{XNOR}(x_1, x_2)$ through the constraint applications $\{\mathsf{XOR}(x_1, y), \mathsf{XOR}(x_2, y)\}$. It is easy to verify that if a constraint family $\mathcal{F}$ implements functions in a known hard set $\mathcal{F}'$, then $\mathsf{SAT}(\mathcal{F})$ is hard as well.

The next step is to identify conditions under which one constraint family can implement functions in another constraint family. The main idea is to characterize various properties of interest such that absence of a property allows us to identify a witness that certifies *absence* of the property. For example, a constraint is affine if and only if for all satisfying assignments s_1, s_2 and s_3, the assignment $s_1 \oplus s_2 \oplus s_3$ is also satisfying. This property follows from a classical and well-known characterization of affine subspaces in linear algebra. So given a function f that is not affine, we know that there exist assignments s_1, s_2 and s_3 (not necessarily distinct) such that $f(s_1) = f(s_2) = f(s_3) = 1$ and $f(s_1 \oplus s_2 \oplus s_3) = 0$. We can use such a witness to show that any non-affine f always implements some known hard constraint family. Once we have developed a basic toolkit of such implementations, we can use it to obtain classification theorems in a variety of settings.

5 An Example Classification Theorem

We now briefly illustrate the above ideas by presenting a classification theorem for $\#\mathsf{SAT}(\mathcal{F})$, the problem of counting the number of assignments satisfying a given constraint satisfaction problem. We will outline the proof of a FP/#P dichotomy result for this class. (Recall that FP is the class of functions computable in P, and #P is the counting counterpart of NP.) The notion of implementation used here requires an additional property, namely, whenever $f(\vec{x})$ is true, there exists a *unique* assignment to $\vec{y}$ that satisfies all the constraints. We refer to such an implementation as a faithful implementation and it is easy to see that a faithful implementation preserves the number of solutions. Notice that the implementation of the XNOR function described above is a faithful implementation.

First observe that most of the tractable problems for the decision task become hard when it comes to counting. For instance, while the decision problems 2-SAT and Horn-SAT are in P, the corresponding counting problems #2-SAT and #Horn-SAT are #P-complete (see [24]). Counting the number of solutions of a linear system over a finite field is as easy as deciding its consistency via Gaussian elimination. It turns out that this is essentially the only tractable case, as shown in the following theorem.

Theorem 5.1 ([6]) $\#\mathsf{SAT}(\mathcal{F})$ *is in* FP *if* $\mathcal{F}$ *is affine, and* $\#\mathsf{SAT}(\mathcal{F})$ *is* #P*-complete otherwise.*

If every constraint in $\mathcal{F}$ is affine, then an instance of $\#\mathsf{SAT}(\mathcal{F})$ can be viewed as a system of linear equations over GF(2). We can use Gaussian elimination to determine the number of satisfying solutions in polynomial time. On the other hand, to get #P-completeness of non-affine families, we need some characterization of non-affine families. We start with the following simple characterization due to Schaefer

Lemma 5.2 *A constraint is affine if and only if for all satisfying assignments s_1, s_2 and s_3, the assignment $s_1 \oplus s_2 \oplus s_3$ is also satisfying.*

We then use this characterization to get an implementation of one of three basic non-affine constraints, from *any* non-affine constraint.

Lemma 5.3 *If g is a non-affine constraint then g can faithfully implement (i.e., can implement, while preserving the number of satisfying assignments) one of the three constraints $(x \vee y)$, $(\bar{x} \vee y)$ or $(\bar{x} \vee \bar{y})$,*

This faithful implementation allows to reduce one of the problems #-Positive-2SAT, #-Implicative-2SAT or #-Negative-2SAT to $\#\mathsf{SAT}(\mathcal{F})$ while preserving the number of solutions. This concludes the proof since these three problems are known to be #P-complete.

The full proof of Lemma 5.3 is somewhat long and the reader is referred to [7]. However, we sketch here the main idea and in particular show how the analysis of infinitely many non-affine functions g can be reduced to the analysis of finitely many cases.

Proof Sketch of Lemma 5.3: In this case, by Lemma 5.2 above, we know that there exist assignments s_1, s_2, s_3 such that $g(s_1) = g(s_2) = g(s_3) = 1$, but $g(s_1 \oplus s_2 \oplus s_3) = 0$. The following table describes the situation:

									$g()$
s_1	0...0	0...0	0...0	0...0	1...1	1...1	1...1	1...1	1
s_2	0...0	0...0	1...1	1...1	0...0	0...0	1...1	1...1	1
s_3	0...0	1...1	0...0	1...1	0...0	1...1	0...0	1...1	1
$s_1 \oplus s_2 \oplus s_3$	0...0	1...1	1...1	0...0	1...1	0...0	0...0	1...1	0
	$Z...Z$	$a...a$	$b...b$	$c...c$	$d...d$	$e...e$	$f...f$	$O...O$	

The crucial thing to notice in the above picture is that there are only *eight* distinct type of variables. Using replication of variables within any one of the blocks above (as indicated by the bottom row), we see that g implements a function $h(Z, a, b, c, d, e, f, O)$ on eight variables with $h(0,0,0,0,1,1,1,1) = h(0,0,1,1,0,0,1,1) = h(0,1,0,1,0,1,0,1) = 1$ but $h(0,1,1,0,1,0,0,1) = 0$. Since there are only finitely many functions h, it is possible in principle to give a finite proof that in each case h implements one of the binary OR function. Of course the number of cases, 2^{252}, is slightly large to fit the margin of this article, or the monograph [7]. Here we will resort to Fermat's technique to overcome this obstacle (i.e., no proof). However, the reader can find a proof in [7] where the cases are explored systematically, as part of the big picture in the study of implementations, to give a *really* finite proof of this fact.

6 Conclusion

One of the most important reasons to study constraint satisfaction problems is that it provides an excellent platform to search for a "formal basis" for "empirical observations". Statements of the form "Natural problems in complexity class $\mathcal{C}$ exhibit such and such behavior" can be formalized and proven in this setting, assuming one is willing to live with the assertion that constraint satisfaction problems are the only natural ones.

A shortcoming in the study of constraint satisfaction problems is that so far classification results have been restricted to problems over a Boolean domain. As mentioned earlier, the problems in this framework do extend quite naturally to non-Boolean domains. However, the proofs of classification do not; and it is open as to whether the theorems do. Jeavons, Cohen and Gyssens [11] have brought to the fore the link between the algebraic closure properties of the constraints and the complexity of the corresponding constraint satisfaction problems. Further the work of Feder and Vardi [8] suggests that this may be the largest class of problems that may exhibit dichotomy, and further that this class of problems are of significant importance from the syntactic perspective on complexity. A complete classification result of all constraint satisfaction problems over arbitrary domains would bring an extremely satisfying conclusion to this line of work, and it would be nice to see work in this direction.

References

[1] Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, PCPs, and nonapproximability – towards tight results. *SIAM Journal on Computing*, 27(3):804-915, June 1998.

[2] Elmar Böhler, Edith Hemaspaandra, Steffen Reith, and Heribert Vollmer. Equivalence problems for Boolean constraint satisfaction. Technical Report 282, Institut für Informatik, Universität Würzburg, 2001.

[3] Stephen A. Cook. The complexity of theorem-proving procedures. In *Conference Record of Third Annual ACM Symposium on Theory of Computing*, pages 151–158, Shaker Heights, Ohio, 3-5 May 1971.

[4] Nadia Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *Journal of Computer and System Sciences*, 51(3):511–522, December 1995.

[5] Nadia Creignou and Jean-Jacques Hébrard. On generating all satisfying truth assignments of a generalized CNF-formula. *Theoretical Informatics and Applications*, 31(6):499–511, 1997.

[6] Nadia Creignou and Miki Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 125(1):1–12, 25 February 1996.

[7] Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems.* SIAM Press, Philadelphia, USA, March 2001.

[8] Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, January 1998.

[9] Harry B. Hunt III, Madhav V. Marathe, and Richard E. Stearns. Generalized CNF satisfiability problems and non-efficient approximability. *Proceedings of the Ninth Annual Structure in Complexity Theory Conference*, pages 356–366, Amsterdam, The Netherlands, 28 June - 1 July, 1994.

[10] Harry B. Hunt III, Richard E. Stearns, and Madhav V. Marathe. Relational representability, Local reductions, and the Complexity of Generalized Satisfiability Problems. *Manuscript*, 2000.

[11] Peter Jeavons, David Cohen and Marc Gyssens. Closure properties of constraints *Journal of the ACM*, 44(4):527–548, July 1997.

[12] Dimitris Kavvadias and Martha Sideri. The inverse satisfiability problem. *SIAM Journal on Computing*, 28(1):152–163, January 1998.

[13] Sanjeev Khanna and Madhu Sudan. The optimization complexity of constraint satisfaction problems. Stanford University Tech. Note STAN-CS-TN-96-29, 1996.

[14] Sanjeev Khanna, Madhu Sudan, and Luca Trevisan. Constraint satisfaction: The approxima-

bility of minimization problems. In *Proceedings 12th Computational Complexity Conference*, IEEE Computer Society Press, pages 282–296, 1997.

[15] Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David P. Williamson. The approximability of constraint satisfaction problems. *SIAM Journal on Computing* (SICOMP), Vol. 30, No. 6, pp. 1863–1920, 2000.

[16] Sanjeev Khanna, Madhu Sudan, and David P. Williamson. A complete classification of the approximability of maximization problems derived from Boolean constraint satisfaction. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 11-20, El Paso, Texas, 4-6 May 1997.

[17] Richard E. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1):155–171, January 1975.

[18] Leonid A. Levin. Universal'nyĭe perebornyĭe zadachi (Universal search problems : in Russian). *Problemy Peredachi Informatsii*, 9(3):265–266, 1973. A corrected English translation appears in an appendix to Trakhtenbrot [22].

[19] Steffen Reith and Heribert Vollmer. Optimal satisfiability for propositional calculi and constraint satisfaction problems. In *Proceedings 25th International Symposium on Mathematical foundations of Computer Science*, volume 1893 of *Lecture Notes in Computer Science*, pages 640–649. Springer-Verlag, 2000.

[20] Thomas J. Schaefer. The complexity of satisfiability problems. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing*, pages 216–226, San Diego, California, 1-3 May 1978.

[21] Richard E. Stearns and Harry B. Hunt III. An algebraic model for combinatorial problems. *SIAM Journal on Computing*, 25(2):448–476, April 1996.

[22] Boris Trakhtenbrot. A survey of Russian approaches to *Perebor* (brute-force search) algorithms. *Annals of the History of Computing* 6:384-400, 1984.

[23] Luca Trevisan, Gregory B. Sorkin, Madhu Sudan and David P. Williamson. Gadgets, approximation, and linear programming. *SIAM Journal on Computing*, 29(6): 2074–2097, December 2000.

[24] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, August 1979.