The average case complexity of multilevel syllogistic

Jim Cox, Lars Ericson, and Bud Mishra

Abstract

An approach to the problem of developing provably correct programs has been to enrich a theorem prover for Hoare logic with decision procedures for a number of decidable sublanguages of set theory ($EMLS$, MLS, and extensions) and arithmetic ($FPILP$) [Sch77]. Citing results of Goldberg [Gol79] on average case behavior of algorithms for SAT, it was hoped that these decision procedures would perform well on average.

So far, it has been fairly difficult to prove average case NP-hardness under the various definitions ([Lev86], [BDCGL89], [BG91], [Gur91], [VR92], [SY92], [RS93]). We should note that the definitions in the literature haven't yet been standardized. We survey some of the results of the average case analysis of NP-complete problems, and compare the results of Goldberg with more pessimistic results. We prove that $FPILP$, $EMLS$ and related fragments of set theory are NP-average complete, and show that there are simple distributions that will frustrate any algorithm for these decision problems.

1 Introduction

One goal of computer science has been to develop a tool T to aid a programmer in building a program P that satisfies a specification S by helping the programmer build a proof in some logic of programs L that shows that P satisfies S. S typically is a pair of propositions (ϕ, ψ) such that, for an input x to P, $\phi(x) \Rightarrow \psi(P(x))$ when P is defined on x. ϕ is called the *precondition* or *assumption*, and ψ is called the *postcondition* or *assertion*. The problem of finding a suitable logic L of programs and specifications and verification tool T may be generically referred to as the "Floyd-Hoare problem", formulated around 1967 [Flo67, Hoa69].

Around 1977, Davis and Schwartz proposed an extension of the Floyd-Hoare problem in which there are multiple assumptions and assertions, referring to the state of a program as execution passes through different *places* π in the program [DS77, Sch77]. A *placed proposition* is then a pair (ϕ, π), where π is either a line of a program or the name of a function. A placed proposition

(ϕ, π) *holds* when, if execution reaches π and the value of the variables X in P is V, then $\phi(V)$ is valid. A *program with assumptions and assertions* or *praa* is then a triple $R = (P, E, F)$ where the assumptions E and assertions F are sets of placed propositions. The pair (E, F) is the specification for P, and R is *correct* when, for every (ϕ, π) in F, if every assumption in E holds, then (ϕ, π) holds. Davis and Schwartz proposed a logic of programs L for establishing the correctness of praas. Deak in her 1980 PhD thesis showed how this logic could be applied to derive several variants of searching algorithms from a common root algorithm [Dea77, Dea80]. The Davis-Schwartz vision of the ideal verification tool T was one in which the inference rules I of L could be extended by decision procedures for commonly occurring programming language constructs, subject to the restriction that the set $\mathcal{R}$ of correct praas verifiable in L would not change under any extension to I. That is, assuming that L is sound, any extension of L is *stable* and consequently also sound. This approach is called the "correct program technology problem" [Sch78].

Davis and Schwartz proposed that a series of decision procedures be discovered for sublanguages of Zermelo-Frankel set theory corresponding to programming constructs commonly used in the high-level set-oriented programming language SETL [Sny90a, Sny90b]. Two key assumptions were that

1. By replacing long proofs with trivial steps by short proofs with complex steps, it would be cognitively easier for a programmer to assist the verification tool in finding a correctness proof for a praa, and the average time cost of verifying a praa would be reduced.

2. The cost $T_{2\circ 1}$ of verifying the correctness of the *composition* of two praas R_1 and R_2 would often be significantly less than the costs T_1, T_2 of verifying the correctness of R_1 and R_2 individually.

One can construct cases in which this latter assumption does not hold. Therefore the hope that composition is cheap must be accompanied by a far more detailed context than the one that says it is easier to write an application when a library of correct programs is available than when one is not. Depending on how well suited the library is to the task at hand, it can be cheaper to start from scratch, because adherence to a specification and adequacy of the specification to express a given intent are separate issues. While these issues certainly impact the feasibility of correct program technology, in this paper we will focus on the complexity of the decision problems associated with the technology.

In the 1980's, many approaches to the Floyd-Hoare problem and the correct program technology problem were explored, including some alternatives to the Davis-Schwartz approach such as the RAPTS system of Cai and Paige [CP89], and the Calculus of Constructions of Huet, Coquand and Paulin-Mohring (the literature includes [CH85],[Coq86],[Moh86], [PM88],[Hue88a],[Hue88b],[Hue89]). In all three cases we see a reliance on decision procedures for establishing equivalence of values constructible by composition from some class of functions, relations and types. It is then natural to focus on the problem of improving on the average cost of decision problems that are in general intractable.

In this paper we examine two sublanguages, fragments of set theory [FOS80, Pol87], and integer linear programming [Dav57, Coo71, Coo72, Sho77, CZ93] used in the Davis-Schwartz approach to correct program technology. Several ways of dealing with the seeming intractability of these languages have been explored. For example, we can search for tiny sublanguages whose decision problem has polynomial time complexity in the worst case [Pra77, Sho81]. Alternatively we can show that the average case time complexity of some selection of the instances of the decision problem is of a lower order of magnitude than the worst case for all instances [DP60, Gol79, Fra86, Fra91, PT92]. In particular, the good performance of both the Davis-Putnam procedure and resolution based methods for SAT on random inputs, has been cited as a hopeful sign for deciding sublanguages of set theory.

While some NP-complete problems seem to possess good average case algorithms, there has recently been an attempt to identify languages for which this is not the case. One approach is to show that the decision problem for a given NP-Complete language is also *average case hard in the sense of Levin* [Lev86]. We will use this approach to analyze the average case complexity of the languages considered in this paper.

In addition to proving the NP-average completeness of $EMLS$ and related fragments of set theory, we will prove the somewhat stronger result that these languages are NP-distributional complete for infinitely many simple (linear time rankable) probability distributions. This implies that such distributions will cause any decision algorithm to have poor (super-polynomial) average case behavior, unless nondeterminism and determinism are equally powerful with respect to exponential time.

The paper is organized as follows: section 2 introduces the languages considered in this paper, section 3 reviews some results on the average behavior of the Davis-Putnam procedure and resolution for SAT, section 4 reviews the theory of average case complexity relevant to our analysis, and section 5 presents our results on the average case complexity of the sublanguages of set theory pertinent to correct program technology. A preliminary version of this work appears in Ericson's thesis [Eri94, chapter 5], which the reader may wish to consult for more details on the other issues raised above.

2 Some key intractable sublanguages

We review some of the key sublanguages relied upon by the decidable sublanguages of set theory approach [FOS80, CFOS87] to correct program technology. The languages are in the domains of set theory and integer linear programming.

2.1 Set theory

A key decidable sublanguage in set theory is multi-level syllogistic (MLS), or sets with $=, \neq, \in, \notin, \cup, \cap$ and $\setminus$ (set difference).

Ferro, et al. [FOS80] define the language MLS (multi-level syllogistic) as follows:

$$
\begin{array}{rcl}
SetVariable & \leftarrow & v_0|v_1|v_2\ldots \\
Symbol & \leftarrow & \emptyset|SetVariable \\
SetBinaryOperator & \leftarrow & \cup|\setminus|\cap \\
SetTerm & \leftarrow & Symbol|SetTerm\ SetBinaryOperator\ SetTerm \\
SetBinaryRelator & \leftarrow & \in\ |\ \notin\ |\ =\ |\ \neq \\
SetRelation & \leftarrow & SetTerm\ SetBinaryRelator\ SetTerm \\
Prop & \leftarrow & SetRelation|\neg Prop|Prop \wedge Prop \\
 & | & Prop \vee Prop|Prop \Rightarrow Prop|Prop \equiv Prop
\end{array}
$$

They define a *literal* to be

$$Literal \leftarrow SetRelation|\neg SetRelation$$

We call these MLS literals. Further, if the only set theoretic operator or relation allowed is membership, we call the subset language of MLS so obtained $MLS_{\in}$.

Let p be a $Prop$. They note:

> Our aim is to solve the validity problem. That is, we seek an algorithm to determine whether p is valid or not. This problem easily reduces to the one of testing a conjunction q of literals for satisfiability. In fact, p is valid if and only if $\neg p$ is unsatisfiable; moreover, $\neg p$ can be brought into disjunctive normal form $q_0 \vee \cdots \vee q_n$, and is satisfiable if and only if at least one of the q_i is satisfiable.

They show how to transform any $SetRelation$ into a member of the following language of conjuncts of elementary literals, called *elementary multi-level syllogistic* or $EMLS$:

$$
\begin{array}{rcl}
SetVariable & \leftarrow & v_0|v_1|v_2\ldots \\
Symbol & \leftarrow & \emptyset|SetVariable \\
ElementaryLiteral & \leftarrow & Symbol = SetVariable \cup SetVariable \\
 & | & Symbol = SetVariable \setminus SetVariable \\
 & | & Symbol = SetVariable \cap SetVariable \\
 & | & Symbol \in SetVariable \\
 & | & SetVariable \notin SetVariable \\
 & | & Symbol = SetVariable
\end{array}
$$

$$
\begin{aligned}
& \mid \quad Symbol \neq SetVariable \\
SetRelation \quad & \leftarrow \quad ElementaryLiteral | ElementaryLiteral \wedge SetRelation
\end{aligned}
$$

If an elementary literal does not use $\neq$, $\setminus$, and $\notin$ we shall call it an *elementary positive literal*.

The primary algorithm proposed to decide $EMLS$ has been the model graph algorithm, and the hopes were that, augmented with suitable heuristics, it would perform well on average. In the worst case, an upper bound on the number of model graphs that need to be checked is 2^{4n^3} where n is the number of variables in the sentence. Coincidentally, the number of distinct $EMLS$ conjuncts is $2^{2n^3+4n^2}$, whose exponent is of the same order of magnitude. Policriti [Pol87] originally proves the NP-completeness of MLS. The problem of finding a satisfying assignment for sentences in $MLS_{\in}$ is NP-complete [COP90].

In this paper, in addition to proving the NP-average completeness of MLS (and related languages), we prove the stronger result that deciding a subset of $EMLS$, that is, conjuncts of elementary positive literals, is both NP-complete and NP-average complete.

2.2 Presburger arithmetic and the feasibility problem for integer linear programming

A key decidable sublanguage within number theory is *Presburger arithmetic* and its purely existentially quantified subcase, the feasibility problem for integer linear programming (call this FPILP). Let the base language of literals be called **ILP**. Presburger arithmetic is **ILP** with unrestricted use of existential and universal quantifiers over integer-valued variables. FPILP is **ILP** with just universal or just existential quantifiers. The grammar of **ILP** is as follows:

$$
\begin{aligned}
\mathbf{ILP} \leftarrow\ & \mathbf{id} = 0 \mid \mathbf{id} = 1 \mid 0 \leq \mathbf{id} \mid \\
& \mathbf{id} = \mathbf{id} + \mathbf{id} \mid \mathbf{id} = \mathbf{id} - \mathbf{id} \mid \\
& \mathbf{ILP} \wedge \mathbf{ILP}
\end{aligned}
$$

According to Oppen [Opp75], by algorithms of Hilbert-Bernays [HB68, pp. 366-375] and Cooper [Coo71, Coo72], it is known that satisfiability of Presburger arithmetic can be solved in no worse than

$$2^{2^{2^n}}$$

time steps for inputs of size n for mixed quantifiers. According to Shostak [Sho77], Cooper's algorithm rarely exhibits the worst case. Oppen has observed that the satisfiability problem for FPILP is NP-complete. The problem of deciding the satisfiability of purely existentially quantified conjuncts ϕ of literals of the form $x < y - c$, $x < c$ or $x = y$ is, on the other hand, solvable in time $O(n^3)$ where n is the number of distinct integer variables occurring in ϕ [Pra77, Sho81].

3 Complexity of SAT algorithms under different sampling distributions

A number of researchers, including [Gol79], [PT92], and [Fra86], have reported good average case performance of algorithms for deciding conjunctive normal form sentences in propositional logic (SAT) on randomly generated formulæ. This has been cited as providing the hope for the good average performance of the model-graph algorithm, augmented with suitable heuristics. Here we consider Goldberg's analysis of the average case of both the Davis-Putnam procedure and resolution based methods for SAT [Gol79], as the other results are similar.

In Goldberg's work on random instances of SAT, a formula with n clauses over r variables is selected by randomly constructing clauses. To construct a clause we flip an unbiased three-sided coin once for each of the r variables. Based on the result of the coin toss, we include the variable in the clause, we include its complement, or it doesn't occur at all. Under this distribution and similar distributions, usually referred to collectively as the constant density model, Goldberg et.al. show that the Davis-Putnam procedure (DPP) has polynomial average time behavior. The results are extended to resolution based methods by showing that a polynomial running time for DPP on an unsatisfiable formula implies a polynomial length resolution refutation.

Chvatal and Szemeredi's paper [CS87] suggests why DPP and resolution have good performance under the constant density model. In their terms, the constant density model sampling distributions select *dense* formulæ with high likelihood; *sparse* instances of SAT are hard for DPP and resolution based methods.

Let $\mathbf{C}(r,k)$ be the set of clauses (disjuncts of literals) containing k literals over r variables. Let $\mathbf{C}'(r,k)$ be the set of clauses in which ℓ and $\neg\ell$ never appear simultaneously in a clause. Let $\mathbf{K}(n,r,k)$ be the n-fold Cartesian product of $\mathbf{C}'(r,k)$. The size of $\mathbf{C}'(r,k)$ is $\binom{r}{k}2^k$. We build a random sentence in $\mathbf{K}(n,r,k)$ by choosing n times from $\mathbf{C}'(r,k)$ with equal likelihood.

The resolution complexity of an unsatisfiable formula is the length of the shortest sequence of clauses constituting a resolution refutation of the formula. Consider the following theorem of Chvatal and Szemeredi [CS87, p. 2]:

Theorem 3.1 *For every choice of positive integers c and k such that $k \geq 3$ and $c2^{-k} \geq 0.7$ there is a positive number ϵ such that, with probability tending to one as n tends to infinity, the random family of cn clauses of size k over n variables is unsatisfiable and its resolution complexity is at least $(1+\epsilon)^n$.*

An immediate consequence of this theorem is that:

Theorem 3.2 *The* average *resolution complexity (and hence a lower bound on the* average *complexity of both DPP and resolution based algorithms) of random sentences in*

$$\mathbf{K}(cr,r,k)$$

with

$$\begin{cases} k \geq 3 \\ \frac{c}{2^k} \geq 0.7 \end{cases}$$

is $\Omega((1+\epsilon)^r)$ *where* $\epsilon > 0$.

This contrasts with Golberg's result. In the Chvatal-Szemeredi result, we are constructing sentences consisting of a large set of small fixed-size clauses, where the number of variables appearing in the sentence is proportionally less than the number of clauses of the sentence, e.g. 3-SAT with at least 6 times as many variables as clauses. On the other hand, Goldberg constructs sentences with large clauses, each variable having a 2/3 probability of appearing in any given clause. Chvatal and Szemeredi are fixing the size of the clauses as the length of the sentences and the number of variables grow (referred to as the constant component size model). In other words, sparse sentences are those where a large number of distinct variables appear in the sentence, yet each clause contains only a small proportion of the total number of variables. While other researchers have constructed infinite families of hard instances that are rare under Goldberg's distribution, Chvatal and Szemeredi have constructed a reasonable sampling distribution where almost all instances are hard.

In conclusion, the way Goldberg, Franco and Policriti-Tetali do their sampling essentially avoids the infinite set of hard instances constructed by Chvatal and Szemeredi. Sparse formulæ with bounded-length clauses are hard for resolution and DPP, dense formulæ are easy. Mitchell et al. (1992) [DML92] point out that the distributions for which known algorithms have average case polynomial time behavior for SAT are somewhat artificial and not the distributions one encounters in Artificial Intelligence applications (e.g. in truth maintenance systems). This brings us to the problem of understanding the average case complexity of SAT.

4 The theory of average case complexity

SAT is, by Cook's Theorem, the canonical NP-complete problem [GJ79]. It also happens to be a problem around which a theory of average case complexity of considerable subtlety has grown ([Lev86], [BDCGL89], [BG91], [Gur91], [VR92], [SY92], [RS93]). This theory, originated around 1984 by the Russian mathematician Leonid Levin, arises from the study of complexity classes. While the work of Goldberg, Franco, Policriti and Tetali, and Chvatal and Szemeredi on the Davis-Putnam procedure has concentrated on the particular algorithm and the effect of a given distribution on that algorithm, the Levin-inspired work focuses on the inherent average case complexity of the problem.

In this section, we:

- Review the rationale for the somewhat non-intuitive definitions of the new theory of average case complexity.

- Give the structure of the theory, sufficient to state results concerning the average case complexity of SAT. In particular, we define distributional problems, average case complexity classes and NP-average completeness.
- State the result that SAT is NP-average complete and give the consequences of this result.
- Give the conditions on reductions needed to prove the NP-average completeness of the languages studied in this paper.

4.1 Rationale for the new theory

The traditional notion of the average case complexity of a machine M on a problem is the average of the time cost of the procedure on all instances of a given size n, weighted by the probability $\mu_n(x)$ of each instance x. We write this as

$$Time_M^{\mu}(n) = \sum_{|x|=n} \mu_n(x)\ time_M(x)$$

where $time_M(x)$ is the actual number of steps needed for M to halt on input x. It turns out that this definition of average case complexity is inadequate. Ben-David *et al.* [BDCGL89] note:

> ...naive formulations of these definitions suffer from serious problems such as not being closed under functional composition of algorithms, being model dependent, encoding dependent etc.

Regarding the functional composition and model-dependence criticism, Gurevich [Gur91] observes that it is easy to find M such that $Time_M^{\mu}(n)$ is polynomial in n, but where, if M computes $f(x)$, and M^2 computes $f(f(x))$, $Time_{M^2}^{\mu}(n)$ is no longer polynomial in n. Reischuk *et.al.* pointed out that there is also a possibility that, since μ is subscripted by n, it could be piecewise-defined so that qualitatively different measures are used for inputs of different sizes.

Such considerations have resulted in a more evolved definition of average-case complexity of a machine. Let $\mu(x)$ be a distribution function defined over the space of all inputs x of any size. Let $T_M(n)$ be such that $Time_M^{\mu}(n) \leq T_M(n)$ for all $n \geq 0$. Then M is called Levin-μ-average T_M-time bounded if

$$\sum_x \mu(x) \frac{T_M^{-1}(time_M(x))}{|x|} \leq 1$$

Example 4.1 Reischuk *et.al.* find that there are still problems with this definition, namely that the functional growth of $\mu(x)$ influences the time bound T [RS93]:

If, for example, one takes the "standard" *uniform probability distribution*, which assigns probability $\mu_{\text{uniform}}(x) := 6/\pi^2 \cdot |x|^{-2} \cdot 2^{-|x|}$ to a string $x \in \{0,1\}^*$ a machine using n^2 steps on every input of length n would already be average $O(n^{1+\epsilon})$-time bounded for arbitrary $\epsilon > 0$.

So instead of Levin-μ-average T_M-time boundedness, they propose a hierarchy of average case complexity classes whose members are *distributional problems* (L, μ) where L is a language and μ is a distribution upon which is imposed, in addition, a restriction on monotone transformations of μ. We will give the more restricted definition and define some of these classes as we proceed. However, the price for this more accurate theory will be a considerable loss in "intuitiveness" over the original, simple definitions.

The intuition underlying Reischuk *et.al.*'s theory of average case complexity is as follows:

- For the space of all inputs, we attach a much higher weight to the inputs we expect to see. This gives us the notion of a distribution function on the set of inputs. We can now define a *distributional problem* to be a decision problem together with the probability distribution function on the instances of that problem, and seek algorithms that have good average case behavior with respect to this distribution function.

- The initial work in average case complexity sought to bound the computational complexity of the distribution function in various ways because distribution functions that were computed using a lot of computational resources could artificially select the hard instances of a problem as being most likely, thereby frustrating a given algorithm. So the idea was to use distribution functions that weren't too complex, either in terms of computing the distribution function or computationally sampling from it.

- The functional growth of the distribution function still affected and led to *poor* estimates of the average case running time of a given algorithm (see Example 4.1 above). By *poor* we mean that our average case complexity determination is not of the same order as the actual performance of our algorithm on the given distribution of inputs.

- In order that the functional growth of the distribution function not influence the upper bound estimate on the average time complexity of a given Turing machine, Reischuk and Schindelhauer bound the complexity of the machine with respect to all monotone transformations on the distribution function. That is, the actual probability assigned to a given input is not primarily important, but only the fact that the input is more (or less) likely than another input. Thus one should be able to "scale" the distribution function while, for each pair of inputs a and b, retaining the fact that a is more likely than b (or vice-versa). This is shown to be equivalent to the notion of the *rankability* of the distribution function. In particular, let the *rank* of an input be the number of elements in the input space of higher probability than a given input, according to the distribution function. One can thus characterize a distribution function based on the amount of resources needed to compute the rank of any given input. Then one defines the average running time of a machine to be total running time averaged over the k most likely inputs (the k inputs of smallest rank). It is this notion that seems to be the "right" one for developing a robust theory of average case complexity.

4.2 Average case complexity classes

Let $\Sigma = \{0, 1\}$. Let $\Sigma^* = \cup_{n \geq 0} \Sigma^n$ be the set of all strings. A *language* is a subset of Σ^*. Let $\text{bin} : \mathcal{N} \to \Sigma^*$ be a standard mapping from natural numbers to a self-delimiting binary encoding that obeys lexicographic order.

In the following, let DTM (respectively, NTM) stand for *deterministic* (respectively, *nondeterministic*) *Turing machine*.

Let $\mu : \Sigma^* \to [0, 1]$ be a probability distribution on strings. Then:

- The *rank* of a problem instance x is the number of problem instances with probability greater than x according to probability measure μ, i.e.

  $$\text{rank}_\mu(x) = |\{z \in \Sigma^* : \mu(z) \geq \mu(x)\}|$$

  This number is always finite, if $\mu(x) > 0$, since if a given instance has probability $p > 0$, then the number of instances that have probability $q \geq p$ is bounded by $\lfloor (1 - p)/p \rfloor$.

- A *complexity bound* is a function $T : \mathcal{N} \to \mathcal{N}$. We assume that any such T is monotone increasing and time-constructible. Time-constructible means that there is a DTM M that on input x runs exactly $T(|x|)$ steps for all $x \in \mathcal{N}$. T is not necessarily injective. We define T^{-1} as it is used below to mean

  $$T^{-1}(m) = \min\{n : T(n) \geq m\}$$

- The *rankability* of a distribution is the amount of effort needed to compute the rank of an input x. Let *T-rankable* be the set of all distributions μ for which there exists a DTM M that on input x computes $\text{bin}(\text{rank}_\mu(x))$ in time $T(|x|)$.

Let POL be the set of complexity bounds

$$POL = \bigcup_{k \in \mathcal{N}} O(N^k)$$

and

$$POL\text{-}rankable = \bigcup_{T \in POL} T\text{-}rankable$$

Hence *POL-rankable* is the class of distributions rankable in polynomial time.

A *distributional problem* is a pair (L, μ) with $L \subseteq \Sigma^*$ a language and $\mu : \Sigma^* \to [0,1]$ a probability distribution.

The class of languages NP are those languages for which membership is decidable in polynomial time on an NTM.

Let $NP^{dist} = \{(L, \mu) : L \in NP \wedge \mu \in POL\text{-}rankable\}$.

A real-valued function $m : [0,1] \to [0,1]$ is called *monotone* if $x \leq y$ implies that $m(x) \leq m(y)$. A real-valued monotone function $m : [0,1] \to [0,1]$ is called a *monotone transformation* of distribution μ if $\sum_x m(\mu(x)) \leq 1$.

Define $Av(T)$ to be the set of pairs (f, μ) such that

$$\begin{aligned} f &: \Sigma^* \to \mathcal{N} \\ \mu &: \Sigma^* \to [0,1] \end{aligned}$$

and for all monotone transformations m,

$$\sum_x m(\mu(x)) \, \frac{T^{-1}(f(x))}{|x|} \leq 1$$

This definition makes reference to all monotone transformations of a distribution function, and is therefore unnecessarily complex. However, by a result of [RS93], a simple alternative definition is that $Av(T)$ consists of all (f, μ) which have the property that if we choose any integer ℓ,

$$\sum_{rank_\mu(x) \leq \ell} \frac{T^{-1}(f(x))}{|x|} \leq \ell$$

In this way, each rank function represents a whole set of distributions. Thus a machine runs in average time T if, for each k, its total running time on the k most likely inputs is no more than kT.

Define $DistDTime(T)$ to be the set of pairs (L, μ) such that

$$(\exists DTM\ M)\ L(M) = L \wedge (time_M, \mu) \in Av(T)$$

Define $AvDTime(T, C)$ to be the set of L such that

$$(\forall \mu \in C)\ (L, \mu) \in DistDTime(T)$$

Define AvP as

$$AvDTime(POL, POL\text{-}rankable)$$

An injective function $f : \Sigma^* \to \Sigma^*$ is a *polynomial time reduction from language* L_1 *to language* L_2 if f can be computed in deterministic polynomial time and $x \in L_1$ if and only if $f(x) \in L_2$.

An injective function $f : \Sigma^* \to \Sigma^*$ is a *distributional reduction* from the distributional problem (L_1, ρ_1) to the distributional problem (L_2, ρ_2) if:

1. f is a polynomial time reduction from L_1 to L_2.

2. *Domination Condition*:

$$(\exists\ c_0, c_1 > 0)\ (\forall\ x \in \Sigma^*)\ rank_{\rho_2}(f(x)) \leq c_0|x|^{c_1} rank_{\rho_1}(x)$$

Remark: The purpose of the domination condition is to preserve polynomial average time computability. Thus if L_2 has an average polynomial time algorithm under distribution ρ_2, this immediately gives an average polynomial time algorithm for (L_1, ρ_1) under the definition of average time given above.

A distributional problem (L, ρ) is *NP-distributional complete* if $(L, \rho) \in NP^{dist}$ and if for all distributional problems in NP^{dist} there exists a distributional reduction to (L, ρ).

A language L is *NP-average complete* if $L \in NP$ and for all $L' \in NP$ and $\rho' \in$ *POL-rankable* there exists a distribution $\rho \in$ *POL-rankable* such that (L', ρ') has a distributional reduction to (L, ρ). One immediately obtains the following important result of [RS93]:

Theorem 4.1 *If (L, ρ) is NP-distributional complete then L is NP-average complete.*

Reischuk *et.al.* draw the picture, shown in Figure 1, of languages L_i whose recognition problem lies within the worst case complexity class NP and the average case complexity class $AvDTime(T, V\text{-}rankable)$, for varying recognition problem worst-case time complexities T and ranking-function worst-case time complexities V. $h(T)$ is an upper bound on the complexity of ranking functions so that the problem still has an efficient average case algorithm and is called the *nose*. This we may define formally as

$$nose(L) = \{(T, V) \in (POL, POL) : L \in AvDTime(T, V\text{-}rankable)\}$$

4.3 The average case complexity of SAT

In the following deterministic (respectively nondeterministic) exponential time refers to the class of languages such that for any language in the class membership can be decided by a DTM (respectively NTM) in time $O(2^{n^k})$, for some constant k (in time exponential to a polynomial).

Reischuk and Schindelhauer (1993) prove that [RS93]:

Theorem 4.2 *There exists a ranking* $\mathrm{rank}_\mu(x)$ *of linear time complexity, to which there correspond infinitely many probability distributions μ, such that the distributional problem (SAT, μ) is NP-distributional complete. Therefore SAT is NP-average complete.*

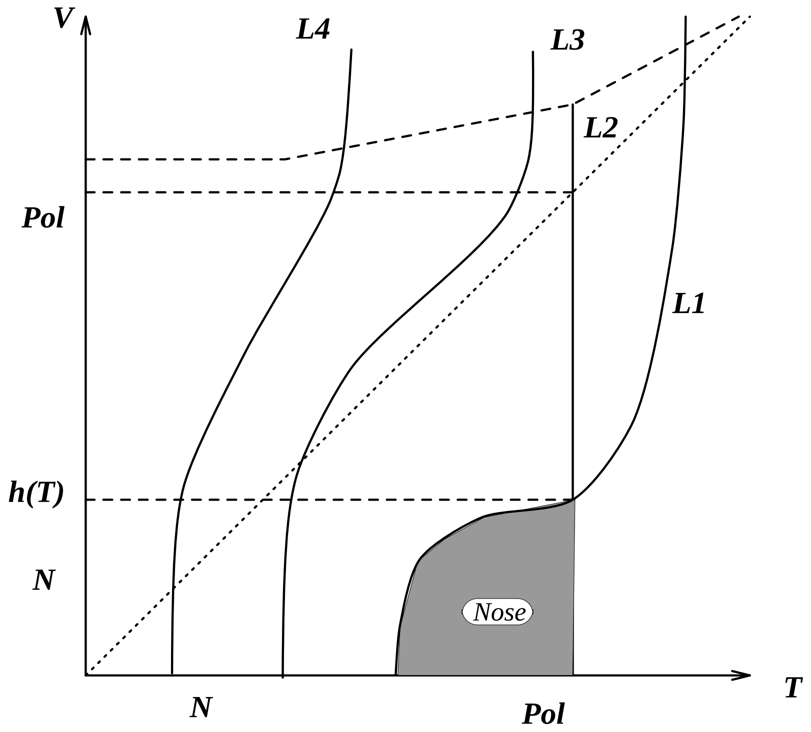

Figure 1: Average Case Complexity of NP-complete Languages.

Proof. (Proof Sketch) Adapting a proof of Levin for the case of computable distributions, the nondeterministic bounded halting problem, $NBH = \{x10^t1^i :$ machine i halts in less than t steps on input $x\}$ is shown to be NP-distributional complete for a distribution with linear time ranking. The ranking is finite only on well-formed instances of NBH and is given by $rank_{NBH}(x10^i1^t) = bin^{-1}(xbin(t)bin(i))$. One can then adapt the standard (Cook's) reduction f of NBH to SAT so that it is linear time invertible. (A technical detail here is to use a low complexity Gödel numbering scheme to map propositional variables with quadruple subscripts to the propositional variables v_1 to v_n.) Then f is a distributional reduction of NBH to SAT with any distribution μ obeying the linear time ranking $rank(x) = rank_{NBH}(f^{-1}(x))$. For example, if the finite ranks of strings are unique, then any distribution μ that assigns probability proportional to $1/p(rank(x)+1)$, where p is a polynomial, will suffice. $\square$

Note that while this ranking of SAT instances may seem somewhat artificial, any ranking such that the rank of an instance x is polynomially related to $rank_{NBH}(f^{-1}(x))$, where f is defined, will do. Under Cook's reduction f, the lengths of x and $f(x)$ are polynomially related, so that it may be possible (but beyond the scope of this paper) to show that SAT is NP-distributional complete for a distribution similar to Chvatal's [CS87].

Reischuk and Schindelhauer note that their results have the following implications:

Theorem 4.3 *If for every linear time rankable distribution there exists an average polynomial time algorithm for SAT then*

- $NP \subseteq AvP$.
- *Every problem in NP can be solved efficiently on average for every polynomial time rankable distribution.*
- *Nondeterministic and deterministic exponential time are equal.*

Proof. The first two consequences are immediate from the previous theorem, i.e., the fact that (SAT, μ) is NP-distributional complete for a linear time rankable distribution μ. The third consequence was originally demonstrated by Ben-David and Luby (see [Gur91]), and the proof uses the standard technique of "padding". That is, given a language L on a binary alphabet in nondeterministic exponential time, one constructs the padded (tally) language $L' = \{1^{[1x]} : x \in L\}$, where $1^{[n]}$ is the unary string of length n, and where $1x$ is the binary string obtained by concatenating 1 with x, regarded as the binary encoding of a natural number. Then L' is in NP. If $NP \subseteq AvP$, then there exists an average polynomial time algorithm A for the padded L' with the specific polynomial time rankable distribution μ, which assigns the probability $6n^{-2}/\pi^2$ to (the unary encoding of) each natural number n, so that the probability is proportional to n^{-2}. Direct calculation shows that the average time and worst case time for this distribution are polynomially related (by a factor of $O(n^3)$), so that algorithm A is a deterministic polynomial time algorithm for L'. This in turn gives a deterministic exponential time algorithm for the original unpadded language L. □

According to Goldberg [Gol79], SAT is decidable in $O(n^3)$ time on average under a very simple distribution. What these results indicate is that there are infinitely many simple (linearly rankable) distributions that will frustrate any algorithm for SAT (e.g. Davis-Putnam), i.e. cause it to have superpolynomial average case behavior. In terms of Figure 1, SAT can have no non-trivial nose.

4.4 Proving NP-average completeness

A polynomial time invertible injective function f is *honest* if $|f^{-1}(y)| \leq r(|y|)$ for some polynomial r. Then the following theorem from [RS93] will enable us to prove the NP-average completeness of the languages studied in this paper:

Theorem 4.4 *Let f be an injective, polynomial time invertible, and honest polynomial time reduction from L_1 to L_2. Then if L_1 is NP-average complete, then so is L_2.*

Proof. (Proof Sketch) Let (L_1, μ) be the distributional problem corresponding to a polynomial time rankable distribution μ. The fact that the reduction f is polynomial time invertible and honest means that the ranking of L_2 given by $\mathrm{rank}_\mu(f^{-1}(x))$, for each x in the range of f, is a polynomial time computable ranking. One can construct a (polynomial time computable) distribution μ', so that $\mathrm{rank}_\mu(f^{-1}(x)) = \mathrm{rank}_{\mu'}(x)$ (one can construct an infinite number of such distributions, for example, any distribution where the probability of x is proportional to $1/p(rank(x)+1)$, for p a polynomial). Thus f provides a distributional reduction from (L_1, μ) to a corresponding (L_2, μ'). Since the choice of polynomial time rankable distribution μ was arbitrary, it follows from the definition that f witnesses the NP-average completeness of L_2. □

5 Average case complexity of correct program technology

5.1 The average case complexity of $EMLS$ and fragments of set theory

With respect to proving the following theorem, we recall reduction f of SAT to $MLS_\in$ [COP90]. Let $\phi \equiv \phi_1 \wedge \cdots \wedge \phi_n$ where each of the ϕ_i is a disjunction of literals (i.e., ϕ is in conjunctive normal form). Let v_i, $i = 1, \ldots, m$ be the distinct propositional variables occurring in ϕ. For each propositional variable we include a corresponding set variable v_i. We also include an additional set variable x. The formula $f(\phi)$ is obtained by substituting the literal $x \in v_i$ for each occurrence of the literal v_i in ϕ, and $x \notin v_i$ for each occurrence of the literal $\neg v_i$ in ϕ.

Theorem 5.1 *The satisfiability problem for $MLS_\in$ is NP-average complete.*

Proof. In order to demonstrate that $MLS_\in$ is NP-average complete, we must show that reduction f of SAT to $MLS_\in$ satisfies the conditions of Theorem 4.4, i.e. that the reduction is injective, polynomial time invertible and honest. Examination of the construction shows that each propositional formula is mapped to a unique $MLS_\in$ formula and that the mapping can be easily inverted. That is, we may determine if the MLS formula is in the range of f by a syntactic check, and if so we output the corresponding propositional formula. Moreover, the length of the $MLS_\in$ instance $f(x)$ is proportional to the length of the original instance of SAT, x. Therefore the reduction is honest, that is, $|x| = |f^{-1}(y)| \leq |y| = |f(x)|$. Since SAT is NP-average complete by Theorem 4.2, by the conditions of Theorem 4.4, $MLS_\in$ is NP-average complete. □

Here we define an extension of MLS. The $(\forall)_0^\ell$*–simple prenex* formulæ ϕ [COP90, Definition 5] are given by the grammar:

$$
\begin{array}{lcl}
\phi & \leftarrow & \bigwedge\limits_{1\le i\le n} \phi_i \\
\phi_i & \leftarrow & (\forall x_{h_1} \in y_{h_1}) \cdots (\forall x_{h_{m_i}} \in y_{h_{m_i}})(L_1^i \vee \cdots \vee L_{k_i}^i) \\
L_i^j & \leftarrow & ElementaryLiteral
\end{array}
$$

with the additional constraints that:

- $m_i > 0$ or $m_i = 0$ and ϕ_i is unquantified.
- The *maximum nesting level* of each variable in every ϕ_i is 1, i.e., in any ϕ_i no x_{h_s} is a y_{h_t} for any s, t.
- The length of the quantifier prefixes in all of the conjuncts ϕ_j does not exceed ℓ.

From the fact that each $MLS_\in$ formula is an instance of of MLS and a quantifier free instance of $(\forall)_0^\ell$*–simple prenex*, we obtain the following corollaries:

Corollary 5.1 *The satisfiability problem for MLS is NP-average complete.*

Corollary 5.2 *The satisfiability problem for* $(\forall)_0^\ell$–simple prenex *is NP-average complete.*

Proof. The reduction f again suffices. The membership of $(\forall)_0^\ell$*–simple prenex* in NP is demonstrated in [COP90]. □

Each of the previous results requires the use of logical disjunction. The following theorem shows that deciding satisfiability of conjunctions of elementary positive literals is equally difficult.

Theorem 5.2 *The satisfiability problem for $EMLS$ is NP-complete and NP-average complete.*

Proof. We first give the reduction f of SAT to a conjunction of MLS literals. We then show how to transform (in linear time) each of the MLS literals into a conjunction of elementary positive literals.

Let $\phi \equiv \phi_1 \wedge \cdots \wedge \phi_n$ where, as above, each of the ϕ_i is a disjunction of literals. Let m be the number of distinct propositional variables v_i occurring in ϕ, and write the literals $\ell_1, \ldots, \ell_{2m}$. Construct $2m$ variables $x_1, \ldots, x_{2m}$ in MLS, and define an association of propositional literals with MLS variables given by the mapping $g(\ell_i)$ such that $g(v_i) \equiv x_i$ and $g(\neg v_i) \equiv x_{i+m}$. Define $\xi \equiv f(\phi)$ to be the MLS formula $\xi_1 \wedge \xi_2$, where

$$\begin{aligned} \xi_1 &\equiv \bigwedge_{1 \le i \le n} (y \in \bigcup_{\ell_j \in \phi_i} g(\ell_j)) \\ \xi_2 &\equiv \bigwedge_{1 \le i \le m} (\emptyset = x_i \cap x_{i+m}) \end{aligned}$$

The idea is that y will be a member of each set that corresponds to a literal assigned "true" in a satisfying assignment. ξ_1 says that each clause must have a set containing y, i.e., a literal assigned "true". ξ_2 says that complementary literals correspond to disjoint sets so that both a literal and its complement cannot be assigned "true". The formal proof that ξ has a model if and only if ϕ is satisfiable is thus straightforward and follows [Pol87]. In particular, a satisfying assignment α for ϕ corresponds to the model M for ξ, where the set $M_y = \emptyset$, and $M_{g(\ell)} \equiv \emptyset$ if $\alpha(\ell) =$ false, else $M_{g(\ell)} \equiv \{\emptyset\}$ (in case $\alpha(\ell) =$ true). Conversely, any model M for ξ provides a satisfying assignment α, where $\alpha(\ell) \equiv$ true if and only if $M_y \in M_{g(\ell)}$.

We can reduce each MLS literal in ξ_1 into a conjunct of elementary literals by the introduction of a linear number of new variables, in the usual way. For example, the MLS literal

$$y \in \bigcup_{1 \le i \le n} v_i$$

becomes

$$(y \in S_1) \wedge (\bigwedge_{1 \le i \le n-1} S_i = S_{i+1} \cup v_i) \wedge (S_n = v_n)$$

The mapping f thus obtained, is an honest, injective, and polynomial time invertible reduction of SAT to $EMLS$. Applying Theorem 4.4, we have that $EMLS$ is NP-average complete (and, of course, NP-complete). □

We can obtain a sharper result.

Corollary 5.3 *There exist infinitely many linear-time rankable probability distributions μ' for which $(EMLS, \mu')$ is NP-distributional complete.*

Proof. Observe that the reduction f from SAT to $EMLS$ is linear time invertible. Thus f provides a distributional reduction from the NP-distributional complete problem (SAT, μ) given in Theorem 4.2, to $(EMLS, \mu')$, where μ' is any probability distribution corresponding to the ranking on $EMLS$ instances y (in the range of f) given by $\mathrm{rank}_\mu(f^{-1}(y))$. Since $\mathrm{rank}_\mu(x)$ is a linear time ranking, then the linear time invertibility of f implies $\text{rank}_{\mu'}(x)$ is also linear time computable. □

As in the case of SAT, a consequence of this proof and Theorem 4.3 is:

Corollary 5.4 *There are infinitely many linear-time rankable probability distributions that will frustrate (cause super-polynomial average time) the model graph algorithm and any other algorithm for $EMLS$, provided that deterministic exponential time is not equal to nondeterministic exponential time.*

Whether these simple distribution functions correspond to the distribution of instances one gets from correct program technology is an open question, although the observations of Section 3 strongly indicate that this is indeed the case.

5.2 The average case complexity of $FPILP$

Theorem 5.3 *$FPILP$ is NP-average complete.*

Proof. Recall the standard reduction of SAT to $FPILP$ [GJ79]. Suppose as above the instance of SAT consists of $\phi \equiv \phi_1 \wedge \cdots \wedge \phi_n$ where each of the ϕ_j is a disjunction of literals, over m propositional variables x_i. The corresponding $FPILP$ instance consists of the constraints

1. $0 \leq x_i \leq 1$, for $1 \leq i \leq m$.
2. Each ϕ_j is represented by the constraint

$$\left(\sum_{x_i \in \phi_j} x_i\right) + \left(\sum_{\neg x_i \in \phi_j} 1 - x_i\right) \geq 1$$

The reduction is injective, invertible by inspection, and honest, since the lengths of the two problems instances are proportional. Thus, $FPILP$ is NP-average complete. □

From the fact that the above reduction is invertible in linear time, we obtain:

Corollary 5.5 *There exist infinitely many linear-time rankable probability distributions μ for which $(FPILP, \mu)$ is NP-distributional complete.*

6 Conclusion

The decidability of many sublanguages of set theory has been extensively investigated. All of these sublanguages are at least NP-complete, which leads us naturally to consider their average-case complexity. We have shown that there are infinitely many simple distributions for which no decision algorithm for these problems can have polynomial time average behavior (unless unlikely complexity theoretic consequences hold).

The natural question to next investigate is what type of distributions of inputs would actually arise in correct program technology practice. One can then examine whether the problems are NP-distributional complete under these distributions. If this turns out to be the case, as we expect, then the results are quite pessimistic, in the sense that theorem provers claiming to decide such languages reasonably fast "on the average" will turn out to be "tuned" with heuristics which, in effect, pre-decide the theorems of interest to the user. That is, whenever such a prover makes recourse to the general algorithm for deciding a sublanguage, it is likely that the prover will run for an unreasonable amount of time. In any case, this research would shed more light on the feasibility of the correct program technology problem.

References

[BDCGL89] Shai Ben-David, Benny Chor, Oded Goldreich, and Michael Luby. On the theory of average case complexity. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 204–216. ACM, 1989.

[BG91] Andreas Blass and Yuri Gurevich. Randomizing reductions of search problems. In *Foundations of Software Technology and Theoretical Computer Science*, pages 10–24. Springer-Verlag Lecture Notes in Computer Science #560, 1991.

[CFOS87] Domenico Cantone, Alfredo Ferro, Eugenio G. Omodeo, and J.T. Schwartz. Decision algorithms for some fragments of anaysis and related areas. *Communications on Pure and Applied Mathematics*, XL:281–300, 1987.

[CH85] Thierry Coquand and Gerard Huet. Constructions: A higher order proof system for mechanizing mathematics. In *EUROCAL '85*. Springer-Verlag Lectures Notes in CS 203, 1985.

[Coo71] D.C. Cooper. Programs for mechanical program verification. In *[MM71]*, pages 43–59, 1971.

[Coo72] D.C. Cooper. Theorem proving in arithmetic without multiplication. In *[MM72]*, pages 91–99, 1972.

[COP90] Domenico Cantone, Eugenio Omodeo, and Alberto Policriti. The automation of syllogistic. II. Optimization and complexity issues. *Journal of Automated Reasoning*, 6(2):173–188, June 1990.

[Coq86] Thierry Coquand. On the analogy between propositions and types. Translation by Walt Hill, HP Labs, of: *Sur l'analogie entre les propositions et les types*, in: **Combinators and Functional Programming Languages**, edited by Guy Cousineau, Pierre-Louis Curien and B. Robinet, Springer-Verlag Lecture Notes in Computer Science 242, 1986.

[CP89] Jiazhen Cai and Robert Paige. Program derivation by fixed point computation. *Science of Computer Programming*, 11:197–261, 1988-1989.

[CS87] Vasek Chvatal and Endre Szemeredi. Many hard examples for resolution. CS Dept, Rutgers University, April 1987.

[CZ93] Edmund Clarke and Xudong Zhao. Analytica: A theorem prover for mathematica. *The Mathematica Journal*, 3(1):56–71, Winter 1993.

[Dav57] Martin D. Davis. A program for Presburger's algorithm. In *Summer Institute for Symbolic Logic*, pages 215–233, Cornell University, Ithaca, NY, 1957.

[Dea77] Edith Gail Deak. Derivation of a related group of searching praas. Technical Report Courant Computer Science Report 12, NYU CS Dept, 1977.

[Dea80] Edith Gail Deak. *A Transformational Approach to the Development and Verification of Programs in a Very High Level Language.* PhD thesis, Computer Science Dept, NYU, June 1980.

[DML92] B. Selman D. Mitchell and H. Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 459–465, 1992.

[DP60] Martin D. Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7(3):201–215, 1960.

[DS77] Martin Davis and Jacob Theodore Schwartz. Metamathematical extensibility for theorem verifiers. Technical Report Courant Computer Science Report 12. Reprinted as [DS79]., NYU CS Dept, 1977.

[DS79] Martin Davis and Jacob Theodore Schwartz. Metamathematical extensibility for theorem verifiers. *Computer and Mathematics with Applications*, 5:217–230, 1979.

[Eri94] Lars Warren Ericson. *Gedanken: A tool for pondering the tractability of correct program technology.* PhD thesis, New York University Computer Science Department, 1994.

[Flo67] R. Floyd. Assigning meanings to programs. *Proc. Symp. Appl. Math.*, 19, 1967.

[FOS80] Alfredo Ferro, Eugenio G. Omodeo, and Jacob T. Schwartz. Decision procedures for elementary sublanguages of set theory, I. Multi-level syllogistic and some extensions. *Communications on Pure and Applied Mathematics*, XXXIII:599–608, 1980.

[Fra86] John Franco. On the probabilistic performance of algorithms for the satisfiability problem. *Information Processing Letters*, 23:103–106, 1986.

[Fra91] John Franco. Elimination of infrequent variables improves average case performance of satisfiability algorithms. *SIAM Journal of Computing*, 20(6):1119–1127, December 1991.

[GJ79] Michael R. Garey and Davis S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., 1979.

[Gol79] Allen T. Goldberg. *On the complexity of the satisfiability problem.* PhD thesis, CS Dept, New York University, October 1979. Courant Institute of Mathematical Sciences Report No. NSO-16.

[Gur91] Yuri Gurevich. Average case completeness. *Journal of Computer and System Sciences*, 42:346–398, 1991.

[HB68] D. Hilbert and P. Bernays. *Grundlagen der Mathematik I.* Springer-Verlag, Berlin, 1968.

[Hoa69] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–583, 1969.

[Hue88a] Gérard Huet. The constructive engine, version 4.5. Projet FORMEL, INRIA, B.P. 105-78153, Le Chesnay Cedex, France, 1988.

[Hue88b] Gérard Huet. Induction principles formalized in the calculus of constructions. In *Programming of Future Generation Computers*, pages 205–215. North-Holland, 1988.

[Hue89] Gérard Huet. *The Calculus of Constructions: Documentation and user's guide version 4.10*, July 1989.

[Lev86] L. Levin. Average case complete problems. *SIAM J. Comput.*, 15:285–286, 1986.

[MM71] B. Meltzer and D. Michie, editors. *Machine Intelligence*, volume 6. American Elsevier, 1971.

[MM72] B. Meltzer and D. Michie, editors. *Machine Intelligence*, volume 7. American Elsevier, 1972.

[Moh86] Christine Mohring. Algorithm development in the calculus of constructions. In *Symposium on Logic in Computer Science*. IEEE Computer Society Press, Cambridge, Massachusetts, 1986.

[Opp75] D. Oppen. *A $2^{2^{2^n}}$ upper bound on the complexity of Presburger arithmetic*. PhD thesis, U. of Toronto, Ontario, 1975.

[PM88] Christine Paulin-Mohring. Extracting f_ω's programs from proofs in the calculus of constructions. Laboratoire d'Informatique, Ecole Normal Supérieure, 45 Rue d'Ulm, 75230 Paris Cedex 05, France, April 28 1988.

[Pol87] Alberto Policriti. The NP-Completeness of MLS. Computer Science Department, New York University, November 1987.

[Pra77] V.R. Pratt. Two easy theories whose combination is hard. Massachusetts Institute of Technology, 1977.

[PT92] Alberto Policriti and Presad Tetali. On the satisfiability problem for the ground case of first order theories. Technical Report Technical Report DIMACS-92-38, Computer Science, Rutgers University, 1992.

[RS93] Rüdiger Reischuk and Christian Schindelhauer. Precise average case complexity. In *10th Annual Symposium on Theoretical Aspects of Computer Science*, pages 650–661, 1993.

[Sch77] Jacob Theodore Schwartz. On correct-program technology. Technical Report Courant Computer Science Report 12, NYU CS Dept, 1977.

[Sch78] Jacob Theodore Schwartz. Program proof technology. Technical Report Technical Survey No. 1, NYU CS Dept, 1978.

[Sho77] Robert E. Shostak. On the SUP-INF method for proving Presbuger formulas. *Journal of the Association for Computing Machinery*, 24(4):529–543, October 1977.

[Sho81] R. Shostak. Deciding linear inequalities by computing loop residues. *Journal of the Association for Computing Machinery*, 28(4):769–779, 1981.

[Sny90a] W. Kirk Snyder. The SETL2 programming language. Department of Computer Science, New York University, September 1990.

[Sny90b] W. Kirk Snyder. The SETL2 programming language: Update on current developments. Department of Computer Science, New York University, September 1990.

[SY92] Rainer Schuler and Tomoyuki Yamakami. Structural average case complexity. In *Foundations of Software Technology and Theoretical Computer Science*, pages 128–139. Springer-Verlag Lecture Notes in Computer Science #652, 1992.

[VR92] R. Venkatesan and S. Rajagopalan. Average case intractability of matrix and Diophantine problems. In *24th Annual ACM STOC*, pages 632–642, May 1992.