THE COMPLEXITY OF VERY SIMPLE BOOLEAN FORMULAS WITH APPLICATIONS*

H. B. HUNT III† and R. E. STEARNS‡§

Abstract. The concepts of SAT-hardness and SAT-completeness modulo \( \text{npolylog}_n \) time and linear size reducibility, denoted by SAT-hard (\( \text{npolylog}_n \), \( n \)) and SAT-complete (\( \text{npolylog}_n \), \( n \)), respectively, are introduced. Regardless of whether \( P = \text{NP} \) or \( P \neq \text{NP} \), it is shown that intuitively

Each SAT-hard (\( \text{npolylog}_n \), \( n \)) problem requires essentially at least as much deterministic time as, and

Each SAT-complete (\( \text{npolylog}_n \), \( n \)) problem requires essentially the same deterministic time as the satisfiability problem for 3CNF formulas.

It is proved that the \( \leq \), satisfiability, tautology, unique satisfiability, equivalence, and minimization problems are already SAT-complete (\( \text{npolylog}_n \), \( n \)), for very simple Boolean formulas and for very simple systems of Boolean equations. These completeness results are used to characterize the deterministic time complexities of a number of problems for lattices, propositional calculi, combinatorial circuits, finite fields, rings \( Z_k (k \geq 2) \), binary decision diagrams, and monadic single variable program schemes. A number of these hardness results are "best possible.

Key words. complexity, NP-completeness, SAT-completeness, decision problems, Boolean formulas, finite fields, modular arithmetic, binary decision diagrams, program schemes, finite and distributive lattices, fault detection

AMS(MOS) subject classifications. 03G99, 06B99, 06D99, 68Q15, 68M15, 94C10

1. Introduction. We study the deterministic time complexity of computational problems for very simple Boolean formulas and for very simple systems of Boolean equations. In particular, we study the fundamental problems of \( \leq \), satisfiability, tautology, unique satisfiability, equivalence, and minimization. There are two reasons for this study.

First, the problem instances we consider are so simple that they can be expected to be encountered in any application area. In contrast, a result derived from complex problem instances might be dismissed in some application areas on the grounds that the formula instances used in the hardness proof are not of the form encountered in practice. In general, proofs obtained from simple instances are better evidence of hardness than proofs obtained from general instances.

Second, hardness results for them are more easily extended to other problems. For example, we obtain results for very simple monotone formulas (formulas without \textbf{not}) and these results easily generalize to many lattices including all nondegenerate finite lattices.

Although our basic technique is to find reductions from the Satisfiability Problem, we will derive results that are sharper than NP-completeness. The disadvantage of

* Received by the editors April 18, 1988; accepted for publication (in revised form) April 4, 1989. A preliminary version of this paper was presented at the Third Annual Symposium on Theoretical Aspects of Computer Science, Paris, Orsay, France. The symposium was supported by the Association Française des Sciences et Techniques de l'Information, de l'Organisation et des Systèmes (AFCET) and the Gesellschaft für Informatik (GI).
† Computer Science Department, State University of New York, Albany, New York 12222.
‡ The research of this author was supported in part by National Science Foundation grants DCR 84-03014 and DCR 86-03184.
§ The research of the author was supported in part by National Science Foundation grant DCR 83-03932.
merely showing NP-completeness is that, for all $\epsilon > 0$, there are NP-complete problems that can be solved in time $2^{O(n^\epsilon)}$. Even $2^{n^{1/3}}$ algorithms should be considered practical, even though "NP-complete" has become associated with "intractable."

Unless explicitly stated otherwise, a Boolean formula is a well-formed formula made up of parentheses, variables, and the operators and, or, and not. A monotone Boolean formula is a Boolean formula without occurrences of not. A literal is a variable or a complemented variable. A 3CNF formula is the conjunction (ands) of clauses where each clause is the disjunction (ors) of at most three literals. 3DNF formulas are defined analogously.

Henceforth, we abbreviate both the satisfiability problem for 3CNF formulas and the set of satisfiable 3CNF formulas by SAT. The sharper technique we use here is to use reductions from SAT that are npolylogn in time and linear in size (output is linear in input). This leads us to the concepts of SAT-hardness (npolylogn, n) and SAT-completeness (npolylogn, n) introduced in § 2. In § 2 we see that “SAT-complete (npolylogn, n)” means “takes essentially the same deterministic time as the satisfiability problem for 3CNF formulas.”

Our key complexity result obtained here concerns the set of formula pairs $(F, G)$ satisfying $F \leq G$, where $F$ and $G$ are such that

1. No variable occurs more than once in $F$ or more than once in $G$,
2. $F$ is a monotone CNF formula,
3. $G$ is a disjunction of monotone CNF formulas.

We show that this set of formula pairs has essentially the same deterministic time complexity as SAT (i.e., is SAT-complete (npolylogn, n)). As corollaries of this basic result, we characterize the deterministic time complexity of a number of basic problems for all finite nondegenerate lattices. Additional applications are presented to logic, circuit analysis and testing, binary decision diagrams, and monadic single variable program schemes. As one corollary, we prove that the recognition of the set of uniquely satisfiable 3CNF formulas requires “essentially the same deterministic time as” SAT. This problem has been extensively studied in the literature (see [30]).

A brief outline of this paper follows. In § 2 we introduce the concepts of npolylogn time and linear size reducibility, SAT-hardness (npolylogn, n), and SAT-completeness (npolylogn, n). We also show that two important reduction procedures can be performed on npolylogn time and linear size bounded Turing machines. In § 3 we present our main deterministic time complexity results for the $\leq$, satisfiability, tautology, unique satisfiability, equivalence, and minimization problems for very simple Boolean equations and for very simple systems of Boolean equations. Theorem 3.3 and Corollary 3.4 are of special importance to the remainder of the paper. In § 4 we use the results and techniques of §§ 2 and 3 to characterize the deterministic time complexities of a number of basic problems (see Fig. 1 in § 4.1) for each nondegenerate finite lattice. Additional applications are presented to logic and to circuit analysis and testing. In § 5 we use the results and techniques of §§ 2 and 3 to characterize the deterministic time complexities of a number of basic problems for each finite field, each ring $\mathbb{Z}_k (k \geq 2)$, binary decision diagrams, and monadic program schemes.

The remainder of this section consists of definitions, notation, and basic results about complexity theory, lattices, and Boolean algebras used in this paper. We assume that the reader is familiar with the complexity classes P, NP, and coNP, polynomial reducibility, NP-hardness and NP-completeness, and coNP-hardness and NP-completeness; otherwise, see [18]. We denote the set of natural numbers by $\mathbb{N}$. Throughout this paper by “Turing machine,” we mean “multiple-tape Turing machine.”

The following problems for Boolean formulas are considered:
(1) The **problem**, i.e., the problem of determining, for Boolean formulas $F$ and $G$, if $F \leq G$, i.e., if $G$ equals 1 whenever $F$ equals 1.

(2) The **satisfiability problem**, i.e., the problem of determining if a Boolean formula $F$ is satisfiable.

(3) The **tautology problem**, i.e., the problem of determining if a Boolean formula $F$ is a tautology.

(4) The **unique satisfiability problem**, i.e., the problem of determining, for a Boolean formula $F$, if there exists exactly one assignment $v$ of values from $\{0, 1\}$ to the variables of $F$ such that $F$ takes on the value 1 under $v$.

(5) The **equivalence problem**, i.e., the problem of determining, for Boolean formulas $F$ and $G$, if $F$ and $G$ denote the same function.

(6) The **minimization problem**, i.e., the problem of finding, given a Boolean formula $F$, an equivalent Boolean formula $G$ such that the number of occurrences of symbols in $G$ is minimal.

**Theorem 1.1** [15], [18]. The set of tautological 3DNF formulas is coNP-complete; and the set of satisfiable 3CNF formulas is NP-complete.

**Definition 1.2.** An algebraic structure $S$ is a nonempty set $S$, called the **domain** of the structure, together with a nonempty set of operations of various arities on $S$. $S$ is said to be **nondegenerate** if $|S| \geq 2$. $S$ is said to be **finite** if $|S| < \infty$, and in addition if $S$ has only finitely many operations, each of finite arity.

**Definition 1.3.** A **lattice** $S = (S, \lor, \land)$ is an algebraic structure with domain $S$ such that $\lor$ and $\land$ are commutative, associative, and idempotent binary operations on $S$ such that, for all $x, y \in S$, $x \lor (x \land y) = x \land (x \lor y) = x$. A distributive lattice $S = (S, \lor, \land)$ is a lattice such that, for all $x, y, z \in S$, $x \lor (y \land z) = (x \lor y) \land (x \lor z)$ and $x \land (y \lor z) = (x \land y) \lor (x \land z)$. A lattice $S = (S, \lor, \land)$ is **finite** if $|S| < \infty$.

Let $S = (S, \lor, \land)$ be a lattice. Let $\leq$ be the partial order on $S$ defined by $x \leq y$ if and only if $x \lor y = y$. An element $a$ of $S$ such that $a \leq b$ for all $b \in S$ is said to be the **minimal element** on $S$ and is denoted by 0. An element $a$ of $S$ such that $b \leq a$ for all $b \in S$ is said to be the **maximal element** of $S$ and is denoted by 1. Let $S = (S, \lor, \land)$ be a lattice with minimal element 0. An element $b$ of $S$ such that $0 < b$ on $S$ but there exists no $c \in S$ for which $0 < c < b$ on $S$ is said to be an **atom** of $S$. A lattice $S = (S, \lor, \land)$ is said to be a **finite depth lattice** if there exists $k \in \mathbb{N}$ such that

$$x_1 < \cdots < x_k$$

on $S$ implies $l \leq k$.

A **Boolean Algebra** has operators $\land$, $\lor$, and $\sim$ and constants 0 and 1 where $\land$, $\lor$, and $\sim$, behave as set intersection, union, and complement, respectively, 0 behaves as the empty set, and 1 behaves like the universal set. Formal axioms can be found in [1], [7], and [43]. We let $\text{BOOLE}$ be the two-element Boolean algebra of everyday logic. We let $\text{BIN}$ be the two-element distributive lattice, namely, $\text{BOOLE}$ without the negation (or complement) operator.

**Theorem 1.4** [7]. (1) Let $L = (S, \lor, \land)$ be a nondegenerate distributive lattice. Let $F$ and $G$ be formulas on $L$ involving only variables, parentheses, $\lor$, and $\land$. Then, $F \leq G$ on $L$ if and only if $F \leq G$ on $\text{BIN}$; and $F = G$ on $L$ if and only if $F = G$ on $\text{BIN}$.

(2) Let $L = (S, \lor, \land, \sim, 0, 1)$ be a nondegenerate Boolean algebra. Let $F$ and $G$ be formulas on $L$ involving only variables, parentheses, $\lor$, $\land$, $\sim$, 0, and 1. Then, $F \leq G$ on $L$ if and only if $F \leq G$ on $\text{BOOLE}$; and $F = G$ on $L$ if and only if $F = G$ on $\text{BOOLE}$.
formulas involving only variables and operators. The distinction between operators and constants can be blurred by the presence of zero-ary operators (such as 0 and 1 in Boolean algebras). We call formulas with these zero-ary operators “constant-free” since they can be interpreted as formulas independent of the domain.

Restricting ourselves to constant-free formulas does not weaken hardness results since we certainly expect them to be included among formulas encountered in practice. We seek results on constant-free formulas that apply to the class of all algebras with the specified operators. Classes of formulas with domain specific constants can sometimes be harder than constant-free formulas due to the complexity of manipulating constants. The complexity of manipulating constants (i.e., the complexity of arithmetic) is not a topic of this paper.

In the case of finite algebraic structures S, the domain of the structure can be specified by giving distinct names to its elements. The complexity of arithmetic on such a structure S is not an issue, since S’s operators can be specified by tables and have constant cost.

**Definition 1.5.** Let S be an algebraic structure with domain S. By a representation of S, we mean a set of |S| distinct constant symbols denoting the elements of S.

The algebraic structures we use here are Boolean algebras, lattices, logics, and rings, which have standard infix notation for formulas. In general, the results apply to any of the easily parsed formula notations. By the size of a formula F denoted by \|F\|, we mean the number of occurrences of symbols in F, where each occurrence of a variable, operator, constant, or parenthesis is treated as a single occurrence. For example, \|(x_{135} \lor x_{321})\| = 5. The size of an equation or a system of equations is defined analogously. This is the natural measure since variables and constants are the objects on which reductions are defined. When considering the time of a reduction on a Turing machine, however, we will take into account the fact that the infinite variable set must actually be represented by strings on some finite alphabet.

We like to measure time complexity as a function of input size rather than input length. When doing this, we use the symbol \|w\| instead of the traditional n. Thus we use \(L \in \text{DTIME}(F(\|w\|))\) to mean the time required to test string w for membership in L is \(F(\|w\|)\) or fewer Turing machine operations. It is assumed that a reasonably efficient encoding of variables into strings is used when a formula is presented to a Turing machine. Specifically, we assume the length of the Turing machine input is at worst \(O(\|w\| \log \|w\|)\).

Let F be a formula on an algebraic structure S with domain S. Let \(\nu\) be an assignment of values from S to the variables of F. We denote the value taken on by F under \(\nu\) by \(\nu[F]\).

### 2. Preliminary results.
Here we present our hardness concepts and prove their implications for complexity. The objective is to establish stronger relationships than NP-hardness. We close the section with efficient time and size bounded Turing machine algorithms for two basic transformations that serve as subroutines in later sections.

In what follows, let \(\Sigma\) and \(\Delta\) be finite nonempty alphabets; and let \(L\) and \(M\) be languages over \(\Sigma\) and over \(\Delta\), respectively.

**Definition 2.1.** We say that \(L\) is npolylog \(n\) time and linear size reducible to \(M\) if there exists an integer \(k \geq 1\) and a function \(f: \Sigma^* \rightarrow \Delta^*\) computable by an \(O(n(\log n)^k)\) time-bounded deterministic multiple tape Turing machine such that:

(i) For all \(x \in \Sigma^*\), \(x \in L\) if and only if \(f(x) \in M\); and

(ii) There exists \(c > 0\) such that, for all \(x \in \Sigma^*\), \(\|f(x)\| \leq c \cdot \|x\|\).

**Definition 2.2.** We say that \(L\) is SAT-hard (npolylog, \(n\), \(n\)), read “\(L\) is SAT-hard modulo npolylog time and linear size reducibility,” if SAT is npolylog time and...
linear size reducible to $L$ (in which case $L$ is also NP-hard) or unSAT (the set of unsatisfiable 3CNF formulas) is npolylogn time and linear size reducible to $L$ (in which case $L$ is coNP-hard). We say that $L$ is SAT-complete (npolylogn, n), read “$L$ is sat complete modulo npolylogn time and linear size reducibility,” if $L$ is SAT-hard (npolylogn, n), and $L$ is npolylogn time and linear size reducible to either SAT or unSAT.

We will use the term Turing-SAT-complete (npolylogn, n) if the above conditions hold for Turing reductions instead of many-one reductions, where npolylogn is the time spent to create a bounded number of problem instances of linear size.

The basic deterministic time hardness properties of SAT-hard and SAT-complete (npolylogn, n) languages are summarized in Proposition 2.3. Part (1) of the proposition applies if $P = NP$ and says that, in the case of SAT-completeness, the complexity of $L$ and SAT are bounded by the same polynomials. Part (2) applies if the NP-complete problems take exponential time and says, that in the case of SAT-completeness, the complexities of $L$ and SAT have similar polynomials in their exponents. Thus, intuitively, this proposition says that:

(1) If $L$ is SAT-hard (npolylogn, n), then $L$ requires “essentially at least as much deterministic time as SAT,” and

(2) If $L$ is SAT-complete (npolylogn, n), then $L$ requires “essentially the same deterministic time as SAT.”

We note that each SAT-hard (npolylogn, n) language is either NP- or coNP-hard, and that each SAT-complete (npolylogn, n) language is either NP- or coNP-complete.

**Proposition 2.3.** Let $L$ be a language and let $T(n)$ be any increasing function such that, for all $k$, $T(n) \geq n(\log n)^k$ for almost all $n$:

1. If $L$ is SAT-hard (npolylogn, n) and $L \in \text{Dtime}(T(||w||))$, then $\text{SAT} \in \text{Dtime}(T(c||w||))$ for some constant $c$.

2. If $L$ is SAT-complete (npolylogn, n), then $L \in \text{Dtime}(T(c_1||w||))$ for some $c_1$ if and only if $\text{SAT} \in \text{Dtime}(T(c_2||w||))$ for some $c_2$.

(Statement 2 also applies to Turing-SAT-complete (npolylogn, n) problems.)

**Proof.** If $L$ can be done in time $T(||w||)$, then the reduction permits SAT to be done in time $O(npolylogn) + T(O(n))$, which is $O(T(c||w||))$ for some constant $c$. This proves Part (1). If $L$ is complete, there is a corresponding reduction back to SAT and Part (2) is proved. These remarks apply equally well to Turing reductions.

If, as we suspect, SAT requires deterministic time $2^{O(n^k)}$ for some $k$, the SAT-hard(npolylogn, n) problems will also take at least $2^{O(n^k)}$ time. Complete problems will have the property that $L \in \text{Dtime}(2^{O(||w||^k)})$ if and only if $\text{SAT} \in \text{Dtime}(2^{O(||w||^k)})$.

If $P = NP$ and SAT only requires time $O(n^k)$, then the hard problems will also require time $O(n^k)$ and complete problems will have the property that $L \in \text{Dtime}(O(||w||^k))$ if and only if $\text{SAT} \in \text{Dtime}(O(||w||^k))$.

The next proposition is used extensively in this paper.

**Proposition 2.4.** Let $\Theta$ be any nonempty finite set of Boolean operators. Then there exists a constant $c > 0$ and a deterministic $O(npolylogn)$ time-bounded Turing machine $T$ such that, when given a system of Boolean equations $S$ involving operators from $\Theta$ and the constants 0 and 1 as input, $T$ outputs a 3CNF formula $F_S$ such that

1. $||F_S|| \leq c \cdot ||S||$, and

2. The number of satisfying assignments of $F_S$ equals the number of satisfying assignments of $S$.

**Proof.** This reduction can be done by standard techniques using the principles from [6] and standard compiling techniques. We outline the reduction as a sequence of steps with the expectation that the reader can verify that each step can be carried...
out by a Turing machine in the required time and satisfying conditions (1) and (2) of the proposition. In practice, the ideas can be fit into a one-pass algorithm.

Step 1. The system $S$ can be thought of as a list of formula pairs where the two formulas in each pair are to be made equal. Replace each operator occurrence $\Theta$ in the input formulas with a pair $(\Theta, v)$, where $v$ is a variable distinct from the input variables and the other new variables associated with other operator occurrences.

Step 2. Translate the string into a sequence of equations where the left-hand formula has no operators and the right-hand formula has at most one operator. For each pair $(\Theta, v)$ in the input to this step, there will be an equation $v = \Theta(x_1 \cdots x_k)$ where $\Theta$ is $k$-ary and $x_1 \cdots x_k$ are variables or constants representing the operands associated with the occurrence of $\Theta$ in the input. For each formula pair of $S$, the output will have equation $x = y$ where $x$ and $y$ are the variables (or constants) representing the two formulas.

Step 3. For each equation, there is a 3CNF formula that expresses the same Boolean relationship as the equation. The output of the procedure is the conjunction of all these formulas.

Because $\Theta$ is a finite set of Boolean operators, we are dealing with a finite set of transformations of individual equations into 3CNF. Therefore this last step is linear size bounded.

The next proposition asserts the existence of a subroutine for marking variable occurrence quickly on a Turing machine.

Proposition 2.5. There is a deterministic npolylog $n$ time-bound Turing machine that, given a sequence $F$ of symbols and variables as input, replaces each variable $x$ by a pair $(x, k)$ where $k$ is the integer such that $(x, k)$ is the replacement for the $k$th occurrence of $x$ in $F$.

Proof. The set of integers $\{1, 2, \cdots, n\}$, when denoted by their binary numerals, can be sorted deterministically in npolylog $n$ time on a Turing machine using a standard merge-sort algorithm. The Turing machine of the statement of the proposition uses such an npolylog $n$ time sorting algorithm as a subroutine. Let $k$ be the number of variable occurrences in $F$ and let the $i$th occurrence be $x_i$. This machine, given $F$ as input, executes the following five steps:

Step 1. Extract the string $F_1 = (x_1, 1) \cdots (x_k, k)$ from $F$.

Step 2. Sort the pairs in $F_1$ according to the index of these variables (and preserving the original order among occurrences of the same variable). Call the result $F_2$.

Step 3. Make each pair $(x, i)$ of $F_2$ into a triple $(x, i, l)$ where $(x, i)$ is the $l$th occurrence of $x$ in $F$. This can be done in npolylog $n$ time because Step 2 has made the occurrences of $x$ adjacent. Call the result $F_3$.

Step 4. Sort the triples in $F_3$ according to the second component. This restores the variable occurrence to the original order of $F$. Call this result $F_4$.

Step 5. Take the triples from $F_4$ and attach the third component to the corresponding occurrence in $F$. This is the desired output.

We note that, after executing Step 2 of the algorithm immediately above, the Turing machine of the proof can be modified to output in npolylog $n$ time and in order of increasing variable subscript both the variables of $F$ and the numbers of times they occur in $F$.

3. Hard problems for very simple formulas and systems of equations. We study the deterministic time complexities of the $\leq$, satisfiability, unique satisfiability, tautology, equivalence, and minimization problems for Boolean formulas and systems of Boolean equations. More specifically, we describe very simple formulas and systems for which
these problems are hard. In each case, the results are on the boundary of $\mathbf{NP}$ in that the obvious further simplifications result in problems in $\mathbf{P}$. Some of the results, most notably Theorem 3.3, say that two “easy problems” can be combined in simple ways to get problems that are “hard as they can be.”

In the first theorem, the satisfiability problem for 3CNF formulas with $\leq 3$ repetitions per variable is considered. The $\mathbf{NP}$-hardness of this problem is known and is mentioned in [18]. To put this hard problem into the framework of $\mathbf{SAT}$-completeness (npolylog, n), we must show that reductions exist with the required time and size bound. No reduction is cited in [18].

**Theorem 3.1.** The Satisfiability Problem is $\mathbf{SAT}$-complete (npolylog, n) for CNF formulas with $\leq 3$ literals per clause and $\leq 3$ repetitions per variable. The Tautology Problem is $\mathbf{SAT}$-complete (npolylog, n) for DNF formulas with $\leq 3$ literals per term and $\leq 3$ repetitions per variable.

**Proof.** To verify that these two problems are $\mathbf{SAT}$-hard (npolylog, n), it suffices by duality to give an npolylog time and linear size reduction from the 3SAT to 3SAT for CNF formulas with $\leq 3$ repetitions per variable. The following reduction can be used to reduce any Boolean formula $f$ to a Boolean formula $f_2$ such that

(a) No variable occurs more than three times in $f_2$, and

(b) $f$ is in $\mathbf{SAT}$ if and only if $f_2$ is in $\mathbf{SAT}$.

Let $x_1, \ldots, x_n$ be the variables occurring more than one time in $f$. Let $i_1, \ldots, i_n$ be the number of occurrences of $x_1, \ldots, x_n$, respectively, in $f$. For $1 \leq j \leq n$ and $1 \leq k \leq i_j$, let the variables $x_{j,k}$ be distinct variables. Let $f_1$, $F$, and $f_2$ be the Boolean formulas defined as follows:

(i) $f_1$ is the CNF Boolean formula that results from $f$ by replacing, for $1 \leq j \leq n$ and $1 \leq k \leq i_j$, the $k$th occurrence to the variable $x_j$ in $f$ by the variable $x_{j,k}$. Variables appear in $f_1$ only once.

(ii) For $1 \leq j \leq n$, let $g_j = t_{j,1,} \text{ and } \cdots \text{ and } t_{j,i_j}$ where $t_{j,k} = (x_{j,k} \text{ or } (\text{not } x_{j,k+1}))$ for $k < i_j$ and $t_{j,i_j} = (X_{j,1}, \text{ or } (\text{not } x_{j,1}))$. Formula $g_j$ is true if and only if each $t_{j,k}$ is true, which can happen if and only if all the variables with first subscript $j$ have the same value. These variables appear in $g_j$ only twice.

(iii) Let $F$ be the CNF formula $g_1 \text{ and } g_2 \text{ and } \cdots \text{ and } g_n$. Each variable appears in $F$ two times.

(iv) $f_2$ is the Boolean formula $(F \text{ and } f_1)$.

The formula $F$ is true if and only if, for all assignments $v$ of values from $\{0, 1\}$, $\forall x_{i,j} = \forall x_{i,k}$ for all $i, j, k$. Thus, it is easily seen that the formula $f_2$ satisfies the assertions $a$ and $b$ immediately above. Clearly, $\|f_2\| = \|F\| + \|f_1\| + 3 = O(\|f\|)$. Also clearly when $f$ is a CNF formula, so is $f_2$. Finally, by using the deterministic npolylog time-bounded Turing machine of Proposition 2.5 as a subroutine, it is easy to see that the reduction can be carried out on a deterministic npolylog time-bounded Turing machine.

We note that the reduction of the proof of Theorem 3.1 is parsimonious, i.e., preserves the number of satisfying assignments.

There are two obvious ways the satisfiability problem of Theorem 3.1 can be simplified. One is to allow only two literals per clause and the other is to restrict variables to at most two occurrences. By the results of Cook [15] and Tovey [51] respectively, both these problems are in $\mathbf{P}$. The next result shows that we can get hard problems with only two repetitions if we consider formulas more complex than CNF. However, we do not need to go beyond the conjunction of DNF formulas to get problems that are as hard as they can be.
THEOREM 3.2. Consider the set of Boolean formulas $f$ such that

(i) $f$ is a conjunction of DNF formulas:

(ii) Each variable of $f$ occurs exactly once complemented and once uncomplemented.

The satisfiability problem for formulas in this set is $\text{SAT}$-complete ($\text{npolylog n}$, $n$) and $\text{NP}$-hard.

Proof. Let $f$ be a CNF formula with $\leq 3$ literals per clause and $\leq 3$ repetitions per variable. Formula $f$ can be reduced to a simpler problem if some variable appears only uncomplemented. Just replace the variable by the constant 1 and simplify. A similar simplification can be done if a variable appears only complemented. Therefore, without loss of generality, we may further assume the following:

(1) Each variable of $f$ appears both complemented and uncomplemented.

(2) No variable of $f$ occurs twice complemented (by replacing a variable by its complement a varaible that appears twice complemented and hence once uncomplemented can be converted into a once complemented variable).

Let $x_1, \ldots, x_k$ be the variables of $f$ that occur three times in $f$. For $1 \leq j \leq k$, let $y_j$ and $y_j'$ be distinct variables. Let $f'$ be the Boolean formula that results from $f$ by replacing, for $1 \leq j \leq k$,

- The first uncomplemented occurrence of $x_i$ in $f$ by $y_j$,
- The second uncomplemented occurrence of $x_i$ in $f$ by $y_j'$,
- The occurrence of $x_i$ in $f$ by $y_j$ and $y_j'$.

Under this transformation, the clauses of $f$ become DNF formulas and $f'$ is the conjunction of DNF formulas. Thus condition (i) is satisfied and it is easy to see that (ii) is also satisfied. Also, clearly $\|f\| = O(\|f\|)$. We claim the following:

(3.2.1) $f$ is satisfiable if and only if $f'$ is satisfiable.

(3.2.2) $f'$ is constructible from $f$ on a deterministic $\text{npolylog n}$ time-bounded Turing machine.

The correctness of claims (3.2.1) and (3.2.2) implies the theorem.

It is obvious that a satisfying assignment for $f$ can be made into a satisfying assignment for $f'$. Therefore, to prove the correctness of claim (3.2.1), it suffices to show the following:

If there exists an assignment $v$ of values from $\{0, 1\}$ to the variables of $f'$ such that $v[f'] = 1$ and such that $v[y_j] \neq v[y_j']$ for some $j$ with $1 \leq j \leq k$, then there exists an assignment $w$ of values from $\{0, 1\}$ to the variables of $f'$ such that $w[f'] = 1$ and, for $1 \leq j \leq k$, $w[y_j] = w[y_j']$.

For each such assignment $v$, let $w$ be the assignment that is the same as $v$ accept that, for $1 \leq j \leq k$, if $v[y_j] \neq v[y_j']$, then $w[y_j] = w[y_j'] = 1$. Since $f'$ is a Boolean formula monotone in literals, $1 = v[f'] \leq w[f']$. Finally, the correctness of claim (3.2.2) follows from the proof of Proposition 2.5 (using literals instead of variables).

We might imagine, intuitively, that the “hard” problem instances must be constructed in a series of steps, each of which combines problems that are slightly less hard. Our next result shows that such intuition is wrong, and we can construct problems that are as hard as they can be by combining two “easy problems” with a single binary operator. In this case, the easy problems are monotone Boolean formulas that are the disjunctions of CNF formulas and that do not have variables occurring more than once. These are “easy problems” in that they are always satisfiable, are never tautologies, and their solutions can be counted quickly.

THEOREM 3.3. Let $F$ and $G$ be Boolean formulas such that

(i) No variable occurs more than once in $F$ or more than once in $G$.

Downloaded 02/14/19 to 128.143.1.153. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php
(ii) $F$ is a monotone CNF formula,
(iii) $G$ is the disjunction of monotone CNF formulas.

Then, the following problems are SAT-complete (npolylogn, n):

1. Determining if $F \iff G$,
2. Determining if the formula $(F \land (\neg G))$ is satisfiable,
3. Determining if the formula $(G \lor (\neg F))$ is a tautology,
4. Determining if the formula $(F \Rightarrow G)$ is a tautology, and
5. Determining if the system of equations $F \land G = 0$ has a solution.

Problems (2) and (5) are also NP-hard and Problems (1), (3), and (4) are coNP-hard.

The problems remain hard if

(ii') $F$ is the conjunction of monotone DNF formulas, and
(iii') $G$ is a monotone DNF formula.

Proof. For all Boolean formulas $F$ and $G$, the following are obviously equivalent:

1. $F \iff G$,
2. The formula $(F \land (\neg G))$ is not satisfiable,
3. The formula $(G \lor (\neg F))$ is a tautology,
4. The formula $(F \Rightarrow G)$ is a tautology, and
5. The system $F = 1$ and $G = 0$ has no solution.

Thus to prove the theorem, it suffices to prove that the problem of (1) is SAT-complete (npolylogn, n) for monotone Boolean formulas $F$ and $G$ satisfying conditions (i)-(iii) of the theorem.

Proof of (1). To prove SAT-hardness (npolylogn, n) and coNP-hardness, we give an npolylogn time and linear size reduction that maps a formula $f$ monotone in literals to an inequality (\iff) of monotone Boolean formulas such that $f$ is a tautology if and only if the output inequality holds. When applied to formulas that are the disjunction of CNF formulas where each variable appears exactly once complemented and exactly once uncomplemented, the procedure will output $F$ and $G$ satisfying conditions (i)-(iii). Thus by the dual of Theorem 3.2 and the transitivity of npolylogn time and linear size reducibility, we can conclude that the problem of (1) is SAT-hard (npolylogn, n) for formula $F$ and $G$ satisfying conditions (i)-(iii) of the theorem. Given this, the SAT-completeness (npolylogn, n) of the problem follows immediately from Proposition 2.4.

Let $f$ be a Boolean formula monotone in literals. Let $x_1, \ldots, x_n$ be variables occurring in $f$. Let $y_1, \ldots, y_n$ be distinct variables other than $x_1, \ldots, x_n$. Let $f'$ be the monotone Boolean formula that results from $f$ by replacing, for $1 \leq i \leq n$, the occurrences of $(\neg x_i)$ in $f$ by $y_i$. Let $F_n$ be the monotone Boolean formula $(x_i \lor y_i)$ for $1 \leq i \leq n$. Clearly, the formulas $f'$ and $F_n$ are constructible from $f$ deterministically in linear time. Clearly $f'$ and $F$ are monotone. We claim that

\begin{equation}
(3.3.1) \quad f \text{ is a tautology if and only if } F_n \iff f'.
\end{equation}

To prove (3.3.1), assume $\nu$ is an assignment of values from $\{0, 1\}$ to variables $x_1, \ldots, x_n$ such that $\nu[f] = 0$. Consider the assignment $w$ to $x_1, y_1, \ldots, x_n, y_n$, such that $w[x_i] = \nu[x_i]$ and $w[y_i] = \neg \nu[x_i]$ for all $i \leq n$. Clearly, $w[f'] = 0$ since $f$ and $f'$ are identical formulas after their respective substitution of values for variables and literals. Also $w[F_n] = 1$ because $w[x_i \lor y_i] = 1$ for all $i \leq n$ by construction. Therefore $w[F_n] > w[f']$ and $F_n \iff f'$ fails.

To prove (3.3.1) in the reverse direction, consider an assignment $w$ of values from $\{0, 1\}$ to $x_1, y_1, \ldots, x_n, y_n$ such that $w[F_n] = 1$ and $w[f'] = 0$. Consider the assignment $\nu$ from $\{0, 1\}$ to $x_1, \ldots, x_n$ such that $\nu[x_i] = w[x_i]$ for all $i$. A key fact is that this assignment also satisfies $\nu[x_i] \iff w[y_i]$. This fact follows from $w[x_i \lor y_i] = 1$ (because
w[F_n] = 1) and v[x_i] = w[x_i] (by construction). Again consider the formulas f and f' after substitution for variables and literals. The resulting expressions are identical except that certain occurrences of 1 in f' may be 0 in f. (The reverse situation is prevented by the "key fact.") Because the expressions are monotone, w[f'] ≥ v[f]. But since w[f'] = 0, v[f] = 0 and f is not a tautology. Thus (3.3.1) is proved.

Finally, we must verify that no variable is repeated in f' or F_n, when f has variables appearing once complemented and once uncomplemented. But clearly f' has exactly one occurrence of each x_i and exactly one occurrence of each y_i. F_n is constructed to have only single occurrences independently of f.

The statements about NP-hardness and coNP-hardness are evident from the proof. To prove the result for alternative conditions (ii') and (iii'), observe that F =< G implies \neg G =< \neg F. Applying DeMorgan's laws and replacing variables by their complements then gives result (1) and the others follow as above.

Although parts (1)-(5) of Theorem 3.3 are really five ways of saying the same thing, they have different applications. Part (1) is a statement involving only \land, \lor, and \equiv and it can thus be viewed as a statement about distributive lattices. Part (4) can be viewed as a statement about logics without a negation operator. Parts (2) and (3) say Boolean formulas become hard the very first time tractable formulas are combined. Part (5) addresses systems of equations in which the constants 0 and 1 are available.

The "cause" of hardness in Theorem 3.3 is the two levels of or in G allowed by condition (iii) or the two levels of and in G allowed by condition (ii'). If G has only one level of ors and F only one level of ands, F =< G becomes easy, even under the following circumstances:

(a) F is the disjunction of CNF formulas, not necessarily monotone, in which each CNF formula has no repeated variables.
(b) G is the conjunction of DNF formulas, not necessarily monotone, in which each DNF formula has no repeated variables.

To see this, observe \lor\text{CNF}_i \equiv \land \text{DNF}_j if and only if \text{CNF}_i \equiv \text{DNF}_j for all i and j if and only if \neg\text{CNF}_i \lor \text{DNF}_j is a tautology for all i and j. Under DeMorgan's laws, \neg\text{CNF}_i \lor \text{DNF}_j becomes a DNF formula in which no variable occurs more than twice, and the tautology problem for such formula is known to be in P.

The following corollary shows that the equivalence of monotone formulas is also hard in simple cases:

Corollary 3.4. Testing f = g for formula is coNP-hard and SAT-complete (npolylog, n) even if

1. f is a monotone CNF formula, and
2. g is the disjunction of monotone CNF formulas.

Proof. The proof follows from part (1) of Theorem 3.3, since F =< G if and only if F = F \lor G.

The next result extends Theorem 3.3 to questions about unique satisfiability:

Theorem 3.5. The following problems are NP-hard and SAT-hard (npolylogn, n):

1. Determine if a system of two monotone Boolean equations has a unique solution, even if no variable occurs more than three times.
2. Determine if a 3CNF formula has a unique solution, even if no variable occurs more than three times.

Proof. We first show problem (1). Let F and G be monotone Boolean formulas such that no variable occurs more than once in F and more than once in G. Let x_1, \cdots, x_n be the variables occurring in F or in G. Let y_1, \cdots, y_n be n additional
variables. Then, the following are equivalent:

(i) The system of equations

\[ F = 1, \quad G = 0 \]

does not have a solution, and

(ii) The system of equations

\[ (x_1 \land y_1) \lor \cdots \lor (x_n \land y_n) \lor G = 0, \quad F \lor (y_1 \land \cdots \land y_n) = 1 \]

has a unique solution.

To see the equivalence note that \( x_1 = \cdots = x_n = 0 \) and \( y_1 = \cdots = y_n = 1 \) is a solution of the two equations of (ii). Any other solution of the equations of (ii) is a solution of the equations of (i); and any solution of the equations of (i) can be extended to an additional solution of the equations of (ii) by setting \( y_1 = \cdots = y_n = 0 \). Since (ii) can be obtained from (i) in \( \text{npolylog}_n \) time and (i) is \( \text{NP-hard} \) and \( \text{SAT-hard} \) (\( \text{npolylog}_n, n \)) by part (5) of Theorem 3.3, we have part (1) of this theorem. Problem (1) is reduced to Problem (2) by the procedure of Proposition 2.4. (It is easily verified that this procedure preserves the “at most three repetitions” property.)

We next show that the unique satisfiability problems of the previous theorem have “essentially the same hardness” as SAT. In this case we will be using a Turing reduction instead of a many-one reduction so we have a result for Turing-completeness instead of completeness. Actually the reduction is a simple norm 2 truth-table reduction.

**Proposition 3.6.** The problems of Theorem 3.5 are Turing-SAT-complete (\( \text{npolylog}_n, n \)).

**Proof.** We need only consider the unique 3CNF problem (problem (2) of Theorem 3.5) since the first problem has already been efficiently reduced to the second in the proof of Theorem 3.5.

Let \( f \) be a 3CNF formula. Let \( x_1, \cdots, x_n \) be the variables occurring in \( f \). Let \( y_1, \cdots, y_n \) be additional variables. Then, \( f \) is uniquely satisfiable if and only if

\[ f \text{ is satisfiable, and the Boolean formula } f(x_1, \cdots, x_n) \land f(y_1, \cdots, y_n) \land (x_1 \oplus y_1 \lor \cdots \lor x_n \oplus y_n) \text{ is not satisfiable.} \]

Thus unique satisfiability can be solved by solving satisfiability twice. Each of these formulas is linear in the size of the original.

It is already known that unique SAT is \( \text{coNP-hard} \) and can be solved in polynomial time using \( \text{NP} \) twice as an oracle (see [24], [8]). Our proofs imitate some of the past techniques, verifying the time and size of the reductions and applying them to the special case of limited variable occurrences.

Consider the class of Boolean formulas where no variable appears more than twice. We have a polynomial time algorithm that decides whether such a formula has a unique solution. (We provide this algorithm in the Appendix.) Theorem 3.2 thus tells us that this is a class of formulas where satisfiability is \( \text{NP-complete} \) and unique satisfiability is polynomial. Furthermore, if \( P \neq \text{NP} \), there can be no polynomial parsimonious reduction from satisfiable Boolean formulas to this set, for this would contradict Theorem 3.5(2).

Now we consider minimization for very simple Boolean formulas. We show that the problem is “essentially at least as hard as” SAT. Since minimization is not a language recognition problem, this characterization cannot be expressed in terms of SAT-hard (\( \text{npolygon}, n \)). However the principle is the same. Any solution to the minimization problem can be used to solve some SAT-hard (\( \text{npolylog}_n, n \)) problem in essentially the same time.
Theorem 3.7. Consider the problem of finding the minimal Boolean formula equivalent to a monotone formula in which no variable occurs more than twice. Let \( T(n) \) be any increasing function such that, for all \( k \), \( T(n) \geq n(\log n)^k \) for almost all \( n \). Suppose that \( T(\|w\|) \) bounds above the deterministic time complexity of this problem in terms of formula size. Then \( \text{SAT} \in \text{Dtime}(T(c\|w\|)) \) for some constant \( c \).

Proof. We will show how a minimization procedure can be used to solve problem (1) of Theorem 3.3. Let \( F \) and \( G \) be monotone Boolean formulas in which no variable occurs more than once in \( F \) or more than once in \( G \). Let \( z \) be a variable that is not in \( F \) or \( G \) and consider the formula \( H = (F \land z) \lor G \). The value of \( H \) is independent of \( z \) if and only if \( F \leq G \). But the minimum formula for \( H \) will have variable \( z \) if and only if \( H \) depends on \( z \). Therefore \( F \leq G \) can be solved by scanning the minimum formula for \( H \) for the presence of variable \( z \).

We note that the proof of Theorem 3.7 goes through if the statement of the theorem begins “Consider the problem of finding the minimal monotone Boolean formula . . . .” The results in this section are close to the best possible in that further simplifications almost always give problems that are known to be polynomial.

Finally, direct analogues of the theorems in this section hold for Boolean formulas involving operators other than \( \text{and}, \text{or}, \) and \( \text{not} \). The next result lists a number of cases where hard formulas can be constructed using variables which appear no more than twice.

Corollary 3.8. The satisfiability and tautology problems are \( \text{SAT}-\text{complete} \) (\( \text{npolylog} n, n \)) for Boolean formulas \( F \) that involve only variables, parentheses, and one of the following five possibilities:

(i) The \( \text{nand} \) operator \( \mid \) and the constant \( 1 \),
(ii) The \( \text{nor} \) operator \( \downarrow \) and the constant \( 0 \),
(iii) The implication operators \( \Rightarrow \) and the operator \( \text{not} \),
(iv) The implication operator \( \Rightarrow \) and the constant \( 0 \), or
(v) The exclusive or operator \( \oplus \), the \( \text{and} \) operator \( \bullet \), and the constant \( 1 \).

This statement remains true when \( F \) is restricted so that no variable occurs more than two times in \( F \). Furthermore, the \( \leq \) problem is \( \text{SAT}-\text{complete} \) (\( \text{npolylog} n, n \)) for pairs \( F, G \) of such Boolean formulas such that no variable occurs more than once in \( F \) and more than once in \( G \).

Proof. Recall the following logical identities:

(1) \( \text{not} \, a = a \mid 1 = a \downarrow 0 = a \Rightarrow 0 = a \oplus 1 \),
(2) \( a \lor b = (a \mid 1) | (b \mid 1) = (a \downarrow b) \downarrow 0 = \text{not} \, a \Rightarrow b = (a \Rightarrow 0) \Rightarrow b \),
(3) \( a \land b = (a \mid b) \mid 1 = (a \downarrow 0) \downarrow (b \downarrow 0) = \text{not} \, (a \Rightarrow \text{not} \, b) \), and
(4) \( a \Rightarrow b = 1 \oplus [(1 \oplus b) \bullet a] \).

Because the quantities \( a \) and \( b \) appear once on each side of these identities, the identities can be used to linearly transform expressions written with \{\( \text{and}, \text{or}, \text{not} \)} into expressions of the five types described in the corollary. Furthermore, this transformation will preserve the number of occurrences of each variable. The corollary then follows directly from Theorems 3.2 and 3.3.

4. Applications to lattices, logic, and circuits. We use the results and proof techniques of § 3 to show that a number of basic problems are \( \text{SAT}-\text{hard} \) (\( \text{npolylog} n \)) and/or \( \text{SAT}-\text{complete} \) (\( \text{npolylog} n \)) for a wide collection of lattices. These problems include the \( \leq \), equivalence, and minimization problems for formulas, and the satisfiability and unique satisfiability problems for systems of equations. These lattices include all finite, finite-depth, atomic, and distributive lattices. Such lattices appear throughout
discrete mathematics and computer science, especially in logic [36], [43], [44], combinatorics and geometry [2], [7], [53], and the design, analysis, and testing of combina-
tional logic circuits [11], [12], [19]–[21], [38], [46], [50]. Several applications are
presented to logic and to circuit analysis and testing.

4.1. SAT-hard and -complete problems for lattices. We first show that very close
analogues of the complexity results in § 3 for monotone Boolean formulas hold for
each finite lattice.

Theorem 4.1. Let L = (S, v, ∧) be a finite lattice. Let R be a representation of L.
Consider the problems of Fig. 1 for L and R.
(1) Problems 1–10 of Fig. 1 are SAT-complete (npolylog n, n).
(2) Problems 11 and 12 of Fig. 1 are Turing-SAT-complete (npolylog n, n).
(3) Let T(n) be any increasing function such that, for all k, T(n) ≥ n(log n)^k for
almost all n. Suppose that T(∥w∥) bounds above the deterministic time complexity of
Problem 13 of Fig. 1. Then SAT ∈ Dtime (T(∥w∥)) for some constant c.

Proof. Let L and R be as specified in the statement of the theorem. The proof
has two parts.

Part 1. Proof of indicated lower bounds. It suffices to prove that Problems 2, 4,
6, 8, and 11 of Fig. 1 are SAT-hard (npolylog n, n) and that claim (3) of the statement
of the theorem holds for Problem 13 of Fig. 1. Let a ∈ S be an atom of L. Let a be the
constant symbol of denoting the element a. Let F and G be monotone Boolean
formulas such that

No variable occurs more than once in F and more than once in G.

Let F’ and G’ be the formulas on L and R that result from F and from G, respectively,
by replacing

Each occurrence of and by ∧,

Each occurrence of or by ∨, and

Each occurrence of a variable, say x, by (x ∧ a).

1. The ≤-problem for formulas F and G on L and R.
2. Problem 1 restricted to the case where no variable appears more than once in F or more than once in
   G.
3. The equivalence problem for formulas on L and R.
4. Problem 3 restricted to the case where no variable appears more than once in F or twice in G.
5. Determining if a system of equations on L and R has a solution.
6. Problem 5 restricted to the case of two equations in which no variable appears more than twice, once
   in each equation.
7. Determining if a set of equations f_1 = g_1, · · · , f_k = g_k implies an equation f = g on L and R.
8. Problem 7 restricted to the case f_i = c_i implies f = c on L where c_i and c are constants of R and no
   variable occurs more than once in f_i or once in f.
9. Determining if a Boolean combination of equations of the form f = g where f and g are formulas on
   L and R, is satisfiable on L and R.
10. Determining if a Boolean combination of equation of the form f = g, where f and g are formulas on L
    and R, is true for L.
11. Determining if a system of equations on L and R has a unique solution.
12. Problem 11, even if the system has only three equations and no variable occurs more than four times
    in the system.
13. Given a formula F on L and in which no variable occurs more than twice, finding an equivalence
    formula H on L and R of minimal size.

Fig 1. Problems that are hard for finite lattices.
Then, the following are equivalent.

(a) \( F \leq G \).
(b) \( F' \leq G' \) on \( L \).
(c) \( F' \land G' = G' \) on \( L \).
(d) The system of two equations on \( L \) and \( R \)

\[
F' = a \quad \text{and} \quad G' = 0
\]

has no solution.

(e) \( G' = 0 \) implies \( F' = 0 \) on \( L \).

(f) Let \( x_1, \ldots, x_n \) be the variables occurring in \( F \) or in \( G \). Let \( y_1, \ldots, y_n \) be \( n \) additional variables. The system of three equations on \( L \) and \( R \)

\[
((x_1 \lor y_1) \lor \cdots \lor (x_n \lor y_n)) \lor G = 0, \\
F \lor (y_1 \land \cdots \land y_n) = a, \\
(x_1 \lor y_1) \land \cdots \land (x_n \lor y_n) = a
\]

has a unique solution.

(g) Let \( z \) be a variable not occurring in \( F' \) or in \( G' \). Let \( H' \) be the formula \( (F' \land z \land a) \lor G' \). A formula on \( L \) and \( R \) equivalent to \( H' \) of minimal size does not have an occurrence of the variable symbol \( z \) in it.

This equivalence is obtained by arguments closely similar to those of the proofs of the Theorems 3.3, 3.5, 3.7, and Corollary 3.4. To see this, it suffices to observe that

\[
\{ b \land a \mid b \in S \} = \{ 0, a \}
\]

and

Letting \( \lor' \) and \( \land' \) be the restrictions of \( \lor \) and \( \land \) of \( L \), respectively, to \( \{ 0, a \} \), the structures \( (\{ 0, a \}, \lor', \land') \) and \( \text{BIN} \) are isomorphic distributive lattices.

In part (f), the third equation restricts the \( x_i \) and \( y_i \) to \( \{ 0, a \} \). Thus by Theorem 3.3, Problems 2, 4, 6, 8, and 11 of Fig. 1 are each \( \text{SAT} \)-hard (\( \text{npolylogn} \), \( n \)) and claim (3) of the statement of Theorem 4.1 holds.

**Part 2.** Proof of indicated upper bound. To prove the upper bounds on Problems 1-10 of Fig. 1 claimed by the theorem it suffices to prove that Problem 9 of Fig. 1 is \( \text{npolylogn} \) time and linear size reducible to \( \text{SAT} \). The reduction is a fairly direct extension of that of the proof of Proposition 2.4 and is illustrated in Fig. 2. The reduction of equations on \( L \) to \( \text{SAT} \) uses well-known encodings of finite structures into the two-element Boolean algebra. \( \square \)

The \( \leq \), equivalence, and minimization problems for formulas on a finite lattice were shown to be \( \text{coNP} \)-hard in [26]. The reductions used to prove this are highly nonlinear in size. For example, for distributive lattices the reductions are already of size \( \Theta(\|w\|^2) \). For nondistributive lattices, the reductions are significantly less size efficient.

Part (1) of the proof of Theorem 4.1 can be easily generalized so as to apply to many additional lattices as follows. Let \( L = (S, \lor, \land) \) be a lattice with elements \( b, a \in S \) such that \( a \) covers \( b \). Then, \( \{(c \lor b) \land a \mid c \in S\} = \{b, a\} \). Also letting \( \lor' \) and \( \land' \) be the restrictions of \( \lor \) of \( \land \) of \( L \), respectively, to \( \{b, a\} \), the structures \( \{(b, a), \lor', \land'\} \) and \( \text{BIN} \) are isomorphic distributive lattices. Let \( b \) and \( a \) be distinct constant symbols denoting \( b \) and \( a \), respectively. Then, Problems 1-12 of Fig. 1 are \( \text{SAT} \)-hard (\( \text{npolylogn} \), \( n \)) for formulas and for systems of equations on \( L \), where the only allowable constant symbols are \( b \) and \( a \). The minimization problem for such formulas on \( L \) is also "as hard as" \( \text{SAT} \) in the sense of claim (3) of Theorem 4.1.
Boolean combination of equations:
\[(\neg (f_1 = g_1) \lor (f_2 = g_2)) \land ((f_1 = g_3) \lor \neg (f_2 = g_3))\]

3CNF formula for equation \(f = g\):
\[(\bar{v}_i \lor v_j) \land (v_i \lor \bar{v}_j) \land \{3CNF \text{ formula for } f\} \land \{3CNF \text{ formula for } g\}\]

3CNF formula for the Boolean combination of equations:
\[
\{3CNF \text{ formula for } f_1 = g_1\} \land \\
\{3CNF \text{ formula for } f_2 = g_2\} \land \\
\{3CNF \text{ formula for } f_3 = g_3\} \land \\
\{3CNF \text{ formula for } f_2 = g_3\} \land \\
\{3CNF \text{ formula for } w_1 = \neg (v_i = v_j)\} \land \\
\{3CNF \text{ formula for } w_2 = (v_i = v_j)\} \land \\
\{3CNF \text{ formula for } w_3 = (v_i = v_j)\} \land \\
\{3CNF \text{ formula for } w_4 = \neg (v_i = v_j)\} \land \\
\{3CNF \text{ formula for } (w_1 \lor w_2) \land (w_3 \lor w_4)\}
\]

Finally, the generalized satisfiability and tautology problems, for a formula \(F\) on a lattice \(L\) with 0 and 1, are the problems of determining

If there is an assignment \(v\) of values from the domain of \(L\) to the variables of \(F\) such that \(v[F] = 1\),

and if, for all assignments \(v\) of values from the domain of \(L\) to the variables of \(F\),

\[v[F] = 1.\]

Both problems are decidable deterministically in polynomial time, whenever constant expressions on \(L\) can be evaluated deterministically in polynomial time (e.g., \(L\) is a finite lattice). This is easily seen by noting the following. Let \(v_1\) and \(v_2\) be the assignments of values from \(L\) to the variables of a formula \(F\) on \(L\) such that, for all variables \(x\), \(v_1[x] = 0\) and \(v_2[x] = 1\). Then, \(F = 1\) on \(L\) if and only if \(v_1[F] = 1\) on \(L\). Also, there is an assignment \(v\) of values from \(L\) to the variables of \(F\) such that \(v[F] = 1\) if and only if \(v_2[F] = 1\).

4.2. Distributive lattices with an application to logic. By Theorem 1.4 the lower bounds of Theorem 4.1 also hold for each distributive lattice. In particular, Problems 1-4 of Fig. 1 are SAT-complete (npolylogn, n) for constant-free formulas on any distributive lattice. In the next two propositions, we show how each distributive lattice with 1 can naturally be extended so as to have a SAT-hard (npolylogn, n) generalized tautology or generalized satisfiability problem. In the first proposition, the extension is obtained by appending an “implication” operator such that \(A \Rightarrow B\) means “\(B\) is more true than \(A\).” In the second, we append a “negative” operator such that some lattice element represents “not true.”

**Proposition 4.2.** Let \(L' = (S, \lor, \land, \Rightarrow)\) be a nondegenerate algebraic structure such that

(i) The structure \(L = (S, \lor, \land)\) is a lattice;

(ii) There exists 1 in \(S\) such that, for all \(x \in S\), \(x \leq 1\) on \(L\); and

(iii) The operator \(\Rightarrow\) is binary and, for all \(x, y \in S\), \((x \Rightarrow y) = 1\) on \(L'\) if and only if \(x \leq y\) on \(L\).

Then, the set
\[
\{(F, G) \mid \text{F and G are formulas on } L \text{ such that } F \leq G \text{ on } L\}
\]
is linear size reducible to the set
\[ \{ (F \implies G) \mid F \text{ and } G \text{ are formulas on } L; \text{ and } (F \implies G) = 1 \text{ on } L' \}. \]

In particular, if \( L \) is a distributive lattice, then the set
\[ \{ (F \implies G) \mid F \text{ and } G \text{ are constant-free formulas on } L \text{ such that no variable occurs more than once in } F \text{ and more than once in } G; \text{ and } (F \implies G) = 1 \text{ on } L' \} \]
is SAT-complete (n\text{polylog} n, n).

**Proof.** For arbitrary \( L \), the conclusion follows immediately from (iii). For distributive \( L \), the additional conclusion follows from (iii), Theorem 1.4, and Theorem 3.3.

**Proposition 4.3.** Let \( L' = (S, v, \land, ^\sim) \) be a nondegenerate algebraic structure such that
(i) The structure \( L = (S, v, \land) \) is a distributive lattice,
(ii) There exists 1 in \( S \) such that, for all \( x \in S \), \( x \preceq 1 \) on \( L \), and
(iii) The operator \( ^\sim \) is unary, \( ^\sim 1 \neq 1 \) on \( L' \), and there exists \( b \in S \) for which \( ^\sim b = 1 \) on \( L' \).

Then, the set
\[ \{ (F \land (^\sim G)) \mid F \text{ and } G \text{ are constant-free formulas on } L \text{ such that no variable occurs more than once in } F \text{ and more than once in } G; \text{ and there exists an assignment } v \text{ of values from } S \text{ to the variables such that } v[(F \land (^\sim G))] = 1 \text{ on } L' \} \]
is SAT-complete (n\text{polylog} n, n).

**Proof.** By Theorems 1.4 and 3.3, it suffices to prove that

There exists an assignment \( v \) of values from \( S \) to the variables such that \( v[(F \land (^\sim G))] = 1 \text{ on } L' \) if and only if it is not the case that \( F \preceq G \) on \( L \).

The proof consists of two cases.

**Case 1.** If \( F \preceq G \) on \( L \), then \( v[(F \land (^\sim G))] = 1 \text{ on } L' \) implies that \( v[F] = v[G] = v[^\sim G] = 1 \text{ on } L' \), contradicting (iii).

**Case 2.** Suppose it is not the case that \( F \preceq G \) on \( L \). By Theorem 1.4 it is not the case that \( F \preceq G \) on \( \text{BIN} \). Hence, it is not the case that \( F \preceq G \) on the distributive lattice \( \{ b, 1 \}, v', \land' \) where \( v' \) and \( \land' \) are the restrictions of the operators \( v \) and \( \land \), respectively, to \( \{ b, 1 \} \). Thus, there is an assignment \( v \) of values from \( \{ b, 1 \} \), and hence from \( S \), to the variables such that \( v[G] = b \) and \( v[F] = 1 \). Hence, \( v[(F \land (^\sim G))] = 1 \text{ using (iii)}. \)

A number of the lattice-theoretical models of propositional calculi studied in the literature of algebraic logic [7], [43], [44] are known to satisfy the conditions of Propositions 4.2 and/or 4.3 [43]. Thus, there are many formula theories such that Proposition 4.2 implies that the logical validity and/or decision problems are SAT-complete (n\text{polylog} n, n) for simple formulas. These theories include the propositional calculi of classical two-valued logic in the logical theories \( L_1, L_2, L_3 \), and \( L_4 \) in [36], of positive logic [23], of intuitionistic logic [22], the modal logic \( S_4 \) [32], and for \( m \geq 2 \), the \( m \)-valued logic of Post [42]. Intuitively, these theories are in a class of theories where suitable and, or, and implication operators can be defined by suitable formulas and the axioms and theorems evaluate to "true" in all associated models. Formalizing this class of theories is beyond the scope of this paper.

**4.3. Some applications to circuit analysis and testing.** Theorem 3.3 has a number of immediate applications to circuit analysis and testing, including computing signal
probability [41], [40], computing signal reliability [39], determining the testability of stuck-at faults [19], [38], [46], [50], and detecting the presence of static hazards [11], [12], [16]. To apply the theorem, we first give some definitions and observe some equivalences.

Let $F$ and $G$ be Boolean formulas with principle connectives and and or, respectively, and let $z$ be a variable not occurring in $F$ or $G$. (The principle connectives do not really matter but they are drawn as and and or in Fig. 3(a).) Let the combinational circuits $C_1[F, G]$, $C_2[F, G]$, and $C_3[F, G]$ be constructed from fan-out free monotone circuits for $F$ and for $G$ as shown in Fig. 3.

Given a set of variables, we let $\text{eq}$ be the probability distribution on assignments that result when each variable is independently assigned the value 1 with probability one half. For any predicate $P$, we let $\text{pre}_\text{eq}\{P\}$ be the probability that $P$ is true if the

\[ \begin{align*}
\text{(a)} & \\
\text{(b)} & \\
\text{(c)} & 
\end{align*} \]

Fig. 3. Circuits definitions for § 4.3. (a) The circuit $C_1[F, G]$. (b) The circuit $C_2[F, G]$. (c) The circuit $C_3[F, G]$. 
variables in $P$ are assigned values randomly according to distribution eq.

Given the above definitions, the following statements are equivalent:

(i) $F \leq G$.

(ii) $pr_{eq}(F \text{ and } G = 1) = pr_{eq}(F = 1)$.

(iii) $pr_{eq}(F \text{ or } G = 1) = pr_{eq}(G = 1)$.

(iv) $pr_{eq}(G = 1| F = 1) = 1$.

(v) $pr_{eq}(\text{the output of } C_1[F, G] \text{ is correct, when the gate labeled } \alpha \text{ is stuck-at-one and all other gates are correct}) = 1$.

(vi) The gate labeled $\alpha$ in $C_1[F, G]$ is not testable for a stuck-at-one fault.

(vii) The circuit $C_2[F, G]$ does not have a static 0-hazard, when input $z$ is indeterminant.

(viii) The circuit $C_3[F, G]$ does not have a static 1-hazard, when input $z$ is indeterminant.

The equivalence of the first four statements is obvious. The others require some explanation since we are not giving the formal definitions of stuck-at faults and static hazards. The fault detection problem is to determine, by setting circuit inputs and observing circuit outputs, whether a specified circuit gate is performing properly. In Fig. 3(a), we would like to test if the gate labeled $\alpha$ always gives output one (i.e., is stuck at 1) instead of behaving (as it should) like an or-gate. To test this, we must set the variables so that the gate output should be 0 (i.e., $G$ is false) and the output of circuit $G$ is true. This cannot be done if and only if $F \leq G$. With this explanation, the equivalence of (v) and (vi) to (i) should be apparent.

Static hazards are defined formally in terms of a three-valued logic with values 0, $\frac{1}{2}$, 1 where 0 and 1 behave as FALSE and TRUE and $\frac{1}{2}$ behaves as “undetermined.” In Fig. 3(b), making $z$ underdetermined (assigning $z$ value $\frac{1}{2}$) causes (by definition) the output of the and-gate (which is input to the or-gate) to be undetermined. This indeterminacy will pass through the or-gate (by definition) if and only if the other input to the or-gate is 0 or $\frac{1}{2}$. But this can happen for determined assignments to $x_1 \cdots x_n$ if and only if $\neg F$ or $G$ is not a tautology and hence not $F \leq G$. The equivalence of (i) and (vii) should now be apparent and equivalence to (viii) becomes apparent with a dual argument.

From the above equivalences, the following result is immediate.

**Theorem 4.4.** Let $F$ and $G$ be monotone formulas satisfying the conditions of Theorem 3.3. Let the three-level monotone circuit $C_1[F, G]$, and the simple combinatorial circuits $C_2[F, G]$ and $C_3[F, G]$ be constructed from $F$ and $G$ as in Fig. 3. The following problems are SAT-complete (nploygn, n):

1. Determining if $pr_{eq}(F \text{ and } G = 1) = pr_{eq}(F = 1)$,
2. Determining if $pr_{eq}(F \text{ or } G = 1) = pr_{eq}(G = 1)$,
3. Determining if $pr_{eq}(G = 1| F = 1) = 1$,
4. Determining if $pr_{eq}(\text{the output of } C_1[F, G] \text{ is correct, given that the gate labeled } \alpha \text{ is stuck-at-one and all other gates are operating correctly}) = 1$,
5. Determining if the gate labeled $\alpha$ in $C_1[F, G]$ is testable for a stuck-at-one fault,
6. Determining if the circuit $C_2[F, G]$ has a static 0-hazard, when input variable $z$ is indeterminant, and
7. Determining if the circuit $C_3[F, G]$ has a static 1-hazard, when input variable $z$ is indeterminant.

Conclusions (1)–(5) of Theorem 4.4 show that, even when two easy cases are combined, computing signal probability, computing the probabilities of joint or of conditional events, computing signal reliability, and determining the testability of stick-at faults are “as hard as” the satisfiability problem for 3CNF formulas. (Recall
that \( \text{pr}_{eq}(H = 1) \) can be computed deterministically in polynomial time, when \( H \) is a Boolean formula without repeated variables. Also, recall that the testability of single stuck-at faults can be determined deterministically in polynomial time, for combinational circuits without fanout.) Conclusion (5) strengthens the result of [50] that the testability problem for single stuck-at faults is NP-complete for three-level monotone circuits.

Finally, we point out how testing techniques in the literature can be interpreted in our algebraic context. From [46] it can be inferred that

Determining the testability of a multiple stuck-at fault in a combinational circuit is \( \text{npolylogn} \) time and linear size reducible to determining if a system of equations over the four element Boolean algebra has a solution,

and from [12] it can be inferred that

Determining if a combinational circuit has static 0- or 1-hazards, when a particular input variable is indeterminant, is \( \text{npolylogn} \) time and linear size reducible to determining if a system of equations on the three elements DeMorgan lattice \( L_3 \) has a solution.

In both cases, the later problem is SAT-complete \( \text{npolylogn, n} \). Thus, both determining the testability of single stuck-at faults and determining the presence of static 0- and 1-hazards in the very simple combinational circuits of the statement of Theorem 4.4 are “as hard as” the respective problems for arbitrary combinational circuits.

5. Applications to finite fields, modular arithmetic, binary decision diagrams, and program schemes. We use the results and proof techniques of §3 to show that several basic problems are also SAT-hard \( \text{npolylogn, n} \), for finite fields, rings \( \mathbb{Z}_k \ (k \geq 2) \) of integers modulo \( k \), binary decision diagrams (bdd's) [5], and monadic single variable program schemes [35]. Our new results strengthen and simply NP- and coNP-hardness, results, for rings in [29], [9], and [27] and for bdds and monadic single variable program schemes in [25] and [17]. Assuming \( P \neq \text{NP} \), a number of the results obtained are “best” possible.

5.1. Finite fields and modular arithmetic. In [9] the equivalence problem is shown to be coNP-hard, for formulas on each finite field and on each ring \( \mathbb{Z}_k \ (k \geq 2) \). Here, we use Theorem 3.2 to show, for each of these rings, that the equivalence problem is both coNP-hard and SAT-hard \( \text{npolylogn, n} \) for formulas involving only the operations \( +, -, \cdot \), and exponentiation by constants in which no variable occurs more than two times. As a corollary of the proof, we also show, for each of these rings \( R \), that

Determining if a system of equations on \( R \) in which no variable occurs more than once in each equation

is both NP-hard and SAT-hard \( \text{npolylogn, n} \). This last result strengthens results in [27].

---

1 By exponentiation by constants, we mean that we are allowed to denote a formula \( F \cdot \cdot \cdot \cdot \cdot \cdot \cdot F \ (n \geq 2 \ F's) \) by \( F^n \). For such formulas \( F^n \), the number of occurrences of a variable in \( F^n \) equals the number of occurrences of the variable in \( F \) (rather than, \( n \) times the number of occurrences in the variable in \( F \)).
The following problems are both coNP-hard and SAT-hard (npolylog n):

(i) For all finite fields $F$, determining if a formula on $F$, involving only variables, parentheses, $+, -, \cdot$, exponentiation by constants, and one in which no variable occurs more than two times, is equivalent to 0 on $F$.

(ii) For all $k \geq 2$, determining if a formula on the ring $Z_k$, involving only variables, parentheses, $+, -, \cdot$, exponentiation by constants, and one in which no variable occurs more than two times, is equivalent to 0 on $Z_k$.

Proof. (i). Let $S$ be the domain of $F$. Let $k = |S|$. Recall that, for all $a \in S$, $a^{k-1} = 1$, if $a \neq 0$, and $a^k = 0$, if $a = 0$ [34]. Let $f_0, f_1, f_2, f_3$ be the functions on $S$ defined by

$$f_0(a) = a^{k-1}, \quad f_1(a) = 1 - a, \quad f_2(a, b) = 1 - [(1-a) \cdot (1-b)], \quad f_3(a, b) = a \cdot b.$$ 

Let $f_1, f_2, and f_3$ be the restrictions of $f_1, f_2, and f_3$, respectively, to $\{0, 1\}$.

The structure $\langle \{0, 1\}, f_2, f_3, f_1, 0, 1 \rangle$ is isomorphic to the two-element Boolean algebra. Thus, the claim of the theorem for (i) follows by a direct simulation of the proof of Theorem 3.2 by replacing each occurrence of a variable, say $z$, by $z^{k-1}$, each occurrence of $\lor$ by $f_2$, each occurrence of $\land$ by $f_3$, and each occurrence of $\lnot$ by $f_1$. Since the formulas for expressing $f_1(a), f_2(a, b), and f_3(a, b)$ on $F$ have the same number of occurrences of $a$ and of $b$ as the formulas $\neg a, a \lor b, and a \land b$, respectively, this replacement can be accomplished in deterministic linear time.

(ii) The proof follows that of Corollary 4.3 of [9, pp. 897, 898]. Let $S$ be the domain of $Z_k$. There are two cases.

Case 1. $k = p^m$ for some integer $m \geq 1$ where $p$ is a prime. The proof is the same as that for (i) above except that the function $f_0$ on $S$ is defined by $f_0(a) = a^{p^{-1}(p-1)}$.

Note that by Euler’s theorem, $f_0(a) = 1$, if $p \nmid a$, and $f_0(a) = 0$, if $p | a$.

Case 2. $k = p^m n$ where $p \neq 2$ is a prime, $m$ is an integer $\geq 1$, and $p \nmid n$. By the Chinese Remainder Theorem, $Z_k$ is isomorphic to $Z_p^m \times Z_n$, where the isomorphism $I$ is given by, for all $a \in S$, $I(a) = (a_1, a_2)$, where $a_1 = a \mod p^m$ and $a_2 = a \mod n$. Let $A = \{x | 0 \leq x < p^m n, and p | x\}$, and let $\beta = I^{-1}((1, 0))$. Let $f_0$ be the function on $S$ defined by $f_0(a) = (na)^{p^{-1}(p-1)}$. If $a \in A$, then $p$ divides $a$, and hence, $p^m n$ divides $f_0(a)$. Thus, $f_0(a) = 0$. If $a \in S - A$, then gcd $(na, p^m) = 1$. Thus by Euler’s theorem $(na)^{p^{-1}(p-1)} = 1 \mod p^m$. Also, $(na)^{p^{-1}(p-1)} = 0 \mod n$. Thus, $I((na)^{p^{-1}(p-1)}) = (1, 0)$, and hence, $f_0(a) = \beta$. Let $f_1, f_2, and f_3$ be the functions on $S$ defined by

$$f_1(a) = \beta - a, \quad f_2(a, b) = f_1(a + b), \quad f_3(a, b) = \beta - f_2(\beta - a, \beta - b).$$

Since $p \neq 2, 2\beta \in S - A$. Thus, $f_1$ maps $\{0, \beta\}$ to $\{0, \beta\}$ and $f_2$ and $f_3$ map $\{0, \beta\} \times \{0, \beta\}$.

Let $f'_1, f'_2, and f'_3$ be the restrictions of $f_1, f_2, and f_3$, respectively, to $\{0, \beta\}$. Then, the structure $\langle \{0, \beta\}, f'_2, f'_3, f'_1, 0, \beta \rangle$ is isomorphic to the two element Boolean algebra. As in the proof for (i) above, the formulas for $f'_1, f'_2, and f'_3$ do not have repeated variables and a linear time transformation of problems can be accomplished.

Corollary 5.2. Let $R$ be any finite field or ring $Z_k (k \geq 2)$. Then, determining if a system of two equations on $R$ of the form

$$f_1 = c_1, \quad f_2 = c_2$$

has a solution on $R$ is both NP-hard and SAT-hard (npolylog n), where $f_1$ and $f_2$ are formulas on $R$ involving only variables, parentheses, exponentiation by constants, and one in which no variable occurs more than once in $f_1$ and more than once in $f_2$ and $c_1$ and $c_2$ are constant symbols denoting elements of $R$. 

Downloaded 02/14/19 to 128.143.1.153. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php
Proof. In each case, the theorem follows by a direct simulation of the proof of Theorem 3.3 using the replacement given in the proof of Theorem 5.1. □

5.2. Binary decision diagrams and program schemes. The Executability problem (EP) for a class $C$ of program schemes is the problem of determining, given a scheme $S$ in $C$ and a label $\lambda$ of $S$, if there exists an interpretation $I$ of $S$ such that the statement labeled by $\lambda$ in $S$ is executed during the computation of $S$ under $I$. In [25] the EP has been shown to be NP-complete for the class $Sw$ of monadic single variable program schemes without loops consisting only of predicate tests and halt statements. Theorem 3.2 can easily be combined with the proof of [25] to prove the significantly stronger result that the EP is both NP-complete and SAT-hard (npolylogn, n), for the classes of program schemes $S$ in $Sw$ such that no predicate test occurs more than two times in $S$.

One easy and immediate corollary is the following.

THEOREM 5.3. The isomorphism, strong equivalence, weak equivalence, containment, totality, and divergence problems are already SAT-hard (npolylogn, n), for monadic single variable program schemes $S$ such that no predicate test occurs more than two times in $S$.

Any monadic single variable program scheme in which no predicate test occurs more than once is free. Thus Theorem 3.3 yields a simple, immediate, and direct proof of the following result.

THEOREM 5.4. The weak equivalence and containment problems are coNP-complete and are SAT-hard (npolylogn, n), for the free monadic single variable program schemes.

Proof. Let $F$ and $G$ be monotone Boolean formulas, each without repeated variables. As shown in Fig. 4, the monadic single variable program schemes $SF$ and $SG$ can be constructed from $F$ and $G$, respectively, in deterministic npolylogn time. Clearly, the sizes of $SF$ and $SG$ are linearly bounded in the sizes of $F$ and $G$, respectively. Since no variable is repeated in $F$ or in $G$, no predicate test occurs more than once in $SF$ and more than once in $SG$. Thus, $SF$ and $SG$ are both free. Let $SF'$ and $SG'$ be the free monadic single variable program schemes in Fig. 5. It can easily be seen that the following statements are equivalent:

1. $F \equiv G$;
2. For any interpretation $I$ such that the statement labeled $A$ in $SF'$ is executed during the computation of $SF'$ under $I$, the statement labeled $A$ in $SG'$ is executed during the computation of $SG'$ under $I$;
3. $SF'$ is weakly equivalent to $SG'$; and
4. $SF'$ is contained by $SG'$. □

Each monadic single variable program scheme in $Sw$ can also be viewed as a binary decision diagram. Thus a number of strengthened hardness results for bdds can be read off from Theorems 3.2 and 3.3 and the proofs of Theorems 5.3 and 5.4. For example, the following holds.

THEOREM 5.5. The tautology, satisfiability, and equivalence problems are both coNP-complete, and SAT-complete (npolylogn, n) for bdds in which no variable occurs more than two times. Moreover, the $\equiv$ problem is both coNP-complete and SAT-complete (npolylogn, n) for bdds in which no variable occurs more than one time. □

Finally, let $F(x_1, \ldots, x_n)$ be a Boolean formula denoted by a bdd $DF$ in which no variable occurs more than one time along any path. A straight-line program, to compute the value of $p(F = 1)$ from the values of $p(x_i = 1)$ for any independent probability distribution $p$, can be constructed from $DF$ deterministically in polynomial time. Let $F$ and $G$ be two such formulas. Let $p_F$ and $p_G$ be the associated straight-line programs computed from bdds $DF$ and $DG$. Then, $p_F$ and $p_G$ are equivalent for all
FIG. 4. Program schemes for proof of Theorem 5.3.
assignments of values from \( \{x \text{ is a real } | 0 \leq x \leq 1\} \) to their variables if and only if \( F = G \) if and only if \( p_F \) and \( p_G \) are equivalent for all assignments of values from the reals to their variables. Using the RP algorithm in [29] for the Inequivalence Problem for straight-line on infinite integer domains, we obtain an alternative proof for the following theorem from [10].

**Theorem 5.6.** There are RP algorithms for the inequivalence problem for bdds in which no variable occurs more than once along a path and for the strong equivalence problem [35] for free monadic single variable program schemes.

**6. Conclusion.** The concepts of npolylogn time and linear size reducibility, SAT-hard (npolylogn, n), and SAT-completeness (npolylogn, n) have been introduced. Each SAT-hard (npolylogn, n) problem has been shown to require essentially as much deterministic time as SAT; and each SAT-complete (npolylogn, n) problem has been shown to require essentially the same deterministic time as SAT.
Extending our earlier work in [28], we have proved that the \( \leq \), satisfiability, tautology, unique satisfiability, equivalence, and minimization problems are already SAT-complete (npolylogn, n), for very simple Boolean formulas and systems of Boolean equations. In particular in Theorem 3.3, the \( \leq \) problem has been shown to be SAT-complete (npolylogn, n), for very simple monotone Boolean formulas \( F \) and \( G \) such that no variable occurs more than once in \( F \) or more than once in \( G \). This problem, or equivalent variants of it, has been shown to be directly and naturally npolylogn time and linear size reducible to a number of problems for lattices, logics, combinatorial circuits, finite fields, modular arithmetic, monadic single variable program schemes, and binary decision diagrams. Thus, each of these additional problems is also SAT-hard (npolylogn, n).

Assuming \( P \neq NP \), a number of the hardness results of this paper are “best” possible. In [13] it is shown that there is a deterministic polynomial time algorithm to convert a Boolean formula involving only variables, parentheses, the operators or, and, not, and \( \oplus \), and the constants 0 and 1 in which no variable occurs more than once into an equivalent ordered bdd [17]. In [17] the equivalence problem for ordered bdds is shown to be decidable deterministically in polynomial time. Thus, the equivalence problem is also decidable deterministically in polynomial time, for pairs of Boolean formulas \( (F, G) \)

Involving only variables, parentheses, the operators or, and, not, \( \oplus \), \( \Rightarrow \), \( \bot \), and \( = \), and the constants 0 and 1 in which no variable occurs more than once in \( F \) and more than once in \( G \).

(Contrast this with Corollaries 3.4 and 3.8.) Moreover, the satisfiability problem is decidable deterministically in polynomial time, for systems of equations of the form \( F = c \), where \( F \) is such a Boolean formula, \( c \in \{0, 1\} \), and no variable occurs more than once in the system. (Contrast this with Theorem 3.3(5).) For Boolean formulas involving only the operators or, and, and not, more can be said. Namely, two such formulas in negation normal form (i.e., nots are applied only to variables) are equivalent if and only if they are identical up to commutativity and associativity of or and of and [28].

One immediate corollary is that, for all lattices \( L \), the equivalence problem is decidable deterministically in polynomial time for constant-free formulas on \( L \) in which no variable occurs more than once. (Contrast this with Theorem 4.1(1).) Finally, we recall the remark in § 3 that the unique satisfiability result in Theorem 3.5 is best possible in that the same problem for two repetitions can be solved in polynomial time. As noted in § 3, this means (assuming \( P \neq NP \)) that there is no parsimonious reduction of the satisfiability problem for CNF formulas to the satisfiability problem for Boolean formulas in which no variable occurs more than twice.

**Appendix.** The purpose of this Appendix is to prove the following result mentioned in the discussion after Proposition 3.6.

**Theorem.** Let \( L \) be the set of pairs \((S, F)\) such that

1. \( S \) is a set of variables;
2. \( F \) is a Boolean formula using operators \{and, or, not\}, constants \{TRUE, FALSE\}, variables from \( S \), and parentheses;
3. No variable appears more than twice in \( F \);
4. Only one assignment to variables in \( S \) make \( F \) true.

There is a polynomial time algorithm for \( L \).
Set $S$ must be given as part of the problem so that we can represent the case where some variable of $S$ occurs zero times in $F$. If some variable occurs zero times, then $F$ is not uniquely satisfied.

There are certain simplifications that can be applied to any Boolean formula and which preserve properties (3) and (4) of the Theorem. DeMorgan’s laws can be used so that formula $F$ is monotone in literals. Formulas with constants can be simplified to formulas without constants (or to constant formulas). Pairs of the form $(S, x \land F)$ can be simplified to $(S - \{x\}, F_1)$ where $F_1$ is $F$ with TRUE substituted for $x$. Finally, $(S, \bar{x} \land F)$ can be simplified to $(S - \{x\}, F_0)$ where $F_0$ is $F$ with $x$ replaced by FALSE.

The above simplifications can be applied repeatedly until the formula is a constant or a formula of the form

$$(G_1 \lor H_1) \land \cdots \land (G_k \lor H_k)$$

for some $k \geq 1$. The constant FALSE is never satisfiable and the constant TRUE is uniquely satisfiable if and only if the set of variables is empty. We thus only need a polynomial test for formulas of the form $(*)$.

If formula $(*)$ is uniquely satisfiable, there is an assignment that for each $i$, makes $G_i$ or $H_i$ true. We can assume without loss of generality that it is $G_i$, which is true. If some variable in $S$ does not appear in any $G_i$, that variable can be changed without changing any of the $G_i$, and $(*)$ is not uniquely satisfiable. Thus each variable of $S$ appears at least once in some $G_i$, and therefore the variables can appear at most once in the formula

$$(**) \quad H_1 \land \cdots \land H_k,$$

and $(**)$ must have a satisfying assignment. Thus if $(*)$ is uniquely satisfiable, that assignment must make all the $G_i$ and all the $H_i$ true. Changing the value of a variable $x$ in the assignment must make some $G_i \lor H_i$ false, and so $x$ must appear in both $G_i$ and $H_i$. From the above considerations, we conclude $(*)$ is uniquely satisfiable if and only if the following three conditions hold:

(i) All variables of $S$ appear in $(*)$
(ii) For each $i$, $G_i$ and $H_i$ have the same variables
(iii) For each $i$, $G_i \lor H_i$ is uniquely satisfiable.

Conditions (i) and (ii) are easy to test in polynomial time and we get the main result if we can test condition (iii) in polynomial time. Each variable in formulas $H_i$ and $G_i$ occurs only once (by (3) and (ii)) and $G_i \lor H_i$ will have more than one solution if either $G_i$ or $H_i$ contains the operator or. We conclude that condition (iii) is equivalent to the following two conditions:

(iiiia) Each $G_i$ and $H_i$ is the conjunction of literals;
(iiiib) $G_i$ and $H_i$ have the same literals.

Both these conditions can be tested in polynomial time and the result is proved.

REFERENCES


