Satisfiability Problems

Manindra Agrawal *
Dept. of Computer Science
Indian Institute of Technology
Kanpur 208016, India
manindra@iitk.ernet.in

Thomas Thierauf †
Abt. Theoretische Informatik
Universität Ulm
89069 Ulm, Germany
thierauf@informatik.uni-ulm.de

December 14, 2000

Abstract

We investigate the complexity of satisfiability problems that, surprisingly, (to the best of our knowledge) have not yet been considered in the literature.

CNF-SAT, which we also denote by $\wedge$-$\vee$-SAT, asks for an assignment for a given formula that satisfies *all* of its clauses. While $\wedge$-$\vee$-SAT is **NP**-complete,

- what is the complexity of $\oplus$-$\vee$-SAT, where it is asked for an assignment that satisfies an *odd* number of clauses?

We show that $\oplus$-$\vee$-SAT can be solved by a randomized algorithm in polynomial time. The result is extended to $\oplus$-Th-SAT, where we have a threshold instead of an or at the input level.

MaxSAT, which we also denote by Th-$\vee$-SAT, is a generalization of CNF-SAT, and hence **NP**-complete.

- What is the complexity of the analogous optimization version of DNF-SAT, an easy problem, Th-$\wedge$-SAT?

We show that Th-$\wedge$-SAT is **NP**-complete, and the same holds for Th-$\oplus$-SAT, where we have a parity instead of an and at the input level.

1 Introduction

In a seminal paper, Cook [Coo71] showed that the satisfiability problem for Boolean formulas, SAT, is **NP**-complete. From then on, SAT has been reduced to many other **NP** problems, thereby showing them to be **NP**-complete as well (see [GJ79] for a good introduction to NP completeness). In order to facilitate such reductions it has been useful to have the Boolean formulas in certain *normal forms* or to have certain properties. We give some examples. Most notably, the satisfiability problem for formulas in *conjunctive normal form*, CNF-SAT, or even with the restriction to at most 3 literals per clause, 3-CNF-SAT, is **NP**-complete. Other variants

*Research done while visiting the university of Ulm, Germany. Supported in part by an Alexander von Humboldt fellowship.

†Supported in part by DAAD, Acciones Integradas 1995, 322-AI-e-dr.

are based on 3-CNF-SAT are: Not-All-Equal-SAT [Scha78], where it is asked for a satisfying assignment of a given 3-CNF formula that, for any clause, doesn't set all 3 literals to 1. In One-In-Three-SAT [Scha78], for any clause, exactly one literal should be 1. More general, Not-All-Equal-SAT and One-In-Three-SAT are just special cases of the class of generalized satisfiability problems introduced by Schaefer [Scha78] (see also [GJ79], p. 260). Schaefer gives a dichotomy theorem to classify all the properties one can impose on assignments as in the examples above such that the satisfiabilty problem is still **NP**-complete. An analogous theorem for the corresponding counting problems was recently shown in [CH96].

On the other hand, for certain other restrictions, the satisfiability problem becomes easy: for example for Horn formulas it is solvable in polynomial time, **P** (in fact, it is **P**-complete). 2-CNF-SAT, where there are at most 2 literals per clause, can be solved nondeterministically in logarithmic space, **NL** (in fact, it is **NL**-complete). The satisfiability problem for formulas in *disjunctive normal form*, DNF-SAT, can even be solved in logarithmic space, **L**.

CNF-SAT asks for an assignment satisfying *all* of the clauses of a given formula. If not all clauses can be satisfied, the next interesting question is how many clauses can be simultaneously satisfied at maximum. The question of how well this optimization problem can be approximated has attracted much interest due to the new results on probabilistically checkable proofs (PCP). The associated decision version, MaxSAT, has as input a CNF formula F and a number k; we are then asked whether there is some assignment satisfying at least k clauses of F. It is clear that MaxSAT is as least as difficult as CNF-SAT, and hence is **NP**-complete. An interesting point however is that the MaxSAT problem for 2-CNF-SAT, call it 2-MaxSAT, is **NP**-complete as well, even though 2-CNF-SAT is in **P** [GJS76] (see also [Pap94], Theorem 9.2).

The switch from CNF-SAT to MaxSAT can also be seen as follows. Adapting terminology from circuit theory, we call a CNF formula also a and-*over*-or *formula,* or *having an* and-*gate over an* or-*gate*. Correspondingly, CNF-SAT is also denoted by $\wedge$-$\vee$-SAT. Now consider MaxSAT. This is obtained from $\wedge$-$\vee$-SAT by exchanging the and-gate at the output by a threshold-gate, i.e., MaxSAT = Th-$\vee$-SAT.

Let us now analogously replace the or-gate of DNF formulas by a threshold-gate. That is, we ask for the complexity of Th-$\wedge$-SAT. Although we have derived the problem from an easy one, namely DNF-SAT, we show in Section 3 that Th-$\wedge$-SAT is **NP**-complete.

In this paper we also consider satisfiability problems where in addition to threshold gates we allow parity gates. For example, replacing the and-gate of CNF formulas by a parity-gate, we are asking for assignments satisfying an *odd* number of clauses of a given formula. In Section 2 we show that $\oplus$-$\vee$-SAT can be solved by a randomized Las Vegas type algorithm in polynomial time, in symbols: $\oplus$-$\vee$-SAT $\in$ **RP**. Our technique is to arithmetize such formulas which leads to multilinear polynomials over GF(2). Considering these polynomials in a small extension field, GF(2^s), we can then apply the Schwartz-Zippel Theorem to solve the problem by a probabilistic zero test. If one exchanges the remaining or-gate by a more general threshold-gate, an odd number of threshold gates has to be 1 in order to satisfy a parity-over-threshold formula. By extending the algorithm for $\oplus$-$\vee$-SAT, we show that $\oplus$-Th-SAT is still in **RP**. Moreover, our algorithm can be extended to work even for $\oplus$-Th-SAT with *weighted* threshold gates. This has a nice interpretation in terms of a *integer linear programming* problem where it is just required to satisfy an odd number of inequalities instead of all.

Since we are using 4 operations, namely $\{\wedge, \vee, \text{Th}, \oplus\}$, there are 16 ways of building depth two

formulas. However, for most these, the complexity of the corresponding satisfiability problem is easy to see. For example, as for DNF-SAT, having an or at the output level makes the problem trivial. If we take twice the same operation, then clearly $\wedge$-$\wedge$-SAT, $\vee$-$\vee$-SAT, and $\oplus$-$\oplus$-SAT are in **P**. For Th-Th-SAT, note that and and or are specific threshold operations. Therefore Th-Th-SAT is at least as difficult as CNF-SAT, and hence is **NP**-complete. For the same reason this holds also for Th-$\vee$-SAT and $\wedge$-Th-SAT. To keep track for the reader: we classified already 10 of the 16 satisfiability problems in this paragraph.

The complexity of the remaining 6 is not as obvious. We partition them in 2 groups according to the type of operation at the output level. In Section 2 we show that $\oplus$-$\wedge$-SAT, $\oplus$-$\vee$-SAT, and $\oplus$-Th-SAT can be solved by a randomized Las Vegas type algorithm in polynomial time. Therefore these problems are unlikely to be **NP**-complete. In Section 3, we show that $\wedge$-$\oplus$-SAT can be solved in polynomial time, while Th-$\oplus$-SAT and Th-$\wedge$-SAT are **NP**-complete.

2 Parity at the Output

In CNF-SAT we are seeking for an assignment that satisfies *all* the clauses of a formula. Replacing the and-gate by a parity-gate, we ask for an assignment that satisfies an odd number of clauses. Note first that one can do complements with parity by adding a 1 to its inputs, namely $\oplus(x_1, \ldots, x_n) + 1 = \oplus(1, x_1, \ldots, x_n)$. Hence, using de Morgan's laws, we can transform a parity-over-or formula into an equivalent parity-over-and formula and vice versa. Therefore, $\oplus$-$\vee$-SAT is many-one equivalent to $\oplus$-$\wedge$-SAT. Next observe that we can arithmitize any parity-over-and formula $F(x_1, \ldots, x_n)$. Namely, F corresponds to a polynomial $p_F(x_1, \ldots, x_n)$ over GF(2), where parity is interpreted as addition and and as multiplication. An occurence of a variable x_i in F is kept as x_i in p_F, and a negated variable $\overline{x}_i$ is replaced by $(1 + x_i)$. Clearly, we have

$$F \in \oplus\text{-}\wedge\text{-SAT} \quad \Longleftrightarrow \quad p_F \neq 0.$$

Remark 2.1 #**P** *[Val79a] is the class of counting problems, where one wants to compute the number of solutions of* **NP** *problems. For all the known* **NP***-complete problems, the corresponding counting version is* #**P***-complete. This gives rise to the conjecture that if a counting problem is* not #**P***-complete, then its decision version should not be* **NP***-complete. Note however that the counting version of an easy problem can be* #**P***-complete as well. An example is* DNF-SAT *[Val79b].*

Ehrenfeucht and Karpinski [EK89] have considered the problem of counting the number of zeros of multivariate polynomials over GF(2)*, like p_F from above. They have shown that it is* #**P***-complete. Hence we cannot conclude anything from this result for the complexity of the decision version.*

Polynomial p_F is given as a sum of terms, where each term is a product of the form $x_{i_1} \cdots x_{i_k} \cdot (1 + x_{j_1}) \cdots (1 + x_{j_l})$. In order to test whether p_F is the zero-polynomial, we can multiply out each term and write p_F as a sum of monoms. Then p_F is the zero-polynomial if and only in this process all resulting (intermediate) monoms cancel each other. Note however, that the term above results in 2^l monoms, so that the above procedure works in polynomial time only for $l = O(\log |F|)$. In particular, for k-$\oplus$-$\wedge$-SAT, where any clause has at most k literals, we have

Proposition 2.2 *k-$\oplus$-$\wedge$-SAT $\in$ **P**, for any $k > 0$.*

In the case that l is unbounded, no polynomial-time algorithm is known. Below we give a *randomized* polynomial-time algorithm for $\oplus$-$\wedge$-SAT. As a consequence, $\oplus$-$\wedge$-SAT is very unlikely to be **NP**-complete.

Schwartz [Schw80] and Zippel [Zip79] give efficient algorithms that probabilistically check whether a multivariate polynomial is all zero. Below, we state a variant of their main theorem for *multilinear* polynomials that was implicitely shown by Blum, Chandra, and Wegman [BCW80]. An explicit statement, generalized to arbitrary multivariate polynomials can be found in [IM83].

Theorem 2.3 [BCW80, Schw80, Zip79, IM83] *Let $p(x_1, \ldots, x_n)$ be a multilinear polynomial over a field $\mathbf{F}$ that is not the zero polynomial. Let $S \subseteq \mathbf{F}$ with $|S| > 1$. Then there are at least $(|S| - 1)^n$ points $(a_1, \ldots, a_n) \in S^n$ such that $p(a_1, \ldots, a_n) \neq 0$.*

The standard way to apply the theorem for a zero test is quite obvious: fix a large enough subset S of $\mathbf{F}$, for example, $\|S\| = 2n$. Now, for a randomly chosen point $(r_1, \ldots, r_n) \in S^n$ we have $p(r_1, \ldots, r_n) \neq 0$ with probability greater than $((\|S\| - 1)/\|S\|)^n = (1 - (1/2n))^n \geq 1/2$, when $p \neq 0$.

However, we cannot directly apply the theorem that way in our case because, for $\mathbf{F} = \mathrm{GF}(2)$, there are just 2 elements in $\mathbf{F}$. The trick now is to work in a small extension field of GF(2) instead. This method is mentioned by Ibarra and Moran [IM83] and also used for example by Grigoriev *et.al.* [GKS90] in a slightly different setting (to interpolate sparse polynomials).

Theorem 2.4 *$\oplus$-$\wedge$-SAT $\in$ **RP**.*

Proof. Let $F(x_1, \ldots, x_n)$ be a $\oplus$-$\wedge$-SAT formula and let $p_F(x_1, \ldots, x_n)$ be the corresponding polynomial over GF(2) as defined above. Note that p_F is multilinear. Define $s = \lceil \log 2n \rceil$ and consider $\mathrm{GF}(2^s)$. Let $\widehat{p}_F(x_1, \ldots, x_n)$ denote the extension of p_F over $\mathrm{GF}(2^s)$, so that $\widehat{p}_F$ and p_F agree on $\mathrm{GF}(2)^n$. Suppose $\widehat{p}_F$ is not the zero polynomial. Applying Theorem 2.3 with $S = \mathrm{GF}(2)$, we get that $\widehat{p}_F$, and hence p_F, have a nonzero point in GF(2). Hence we can conclude that p_F is the zero polynomial over GF(2) if and only if $\widehat{p}_F$ is the zero polynomial over $\mathrm{GF}(2^s)$.

In order to evaluate $\widehat{p}_F$ at a point in $\mathrm{GF}(2^s)$, we need an irreducible polynomial $\phi(x) \in \mathrm{GF}(2)[x]$ of degree s. (See for example [vdWae70] for background.) Then $\mathrm{GF}(2^s)$ is isomorphic to $\mathrm{GF}(2)[x]/\phi(x)$. Note that there are only $2^{s+1} = 4n$ polynomials of degree s in $\mathrm{GF}(2)[x]$. Berlekamp [Ber70] showed that irreducibility of a polynomial can be checked in polynomial time. Therefore, by cycling through all possible polynomials, one can find an irreducible one in polynomial time.

Now we can apply Theorem 2.3 with $S = \mathrm{GF}(2^s)$. If $\widehat{p}_F$ is not the zero polynomial, we will detect it with probability greater than 1/2. □

Next we consider $\oplus$-Th-SAT which is a generalization of the just solved $\oplus$-$\wedge$-SAT problem. That is, in each clause, there has to be achieved a certain threshold value of 1's for the clause to be true. The question is whether there is an assignment satisfying an odd number of threshold clauses.

The first idea might be to try a similar approach as for $\oplus$- $\wedge$-SAT, where we interpreted a parity-over-and formula as a (compactly written) polynomial over GF(2). The important point was that we could efficiently evaluate such a polynomial at any given point in an appropriate extension field. However, in the case of a parity-over-threshold formula we do not even have such a compact description of the polynomial. Again it is easy to see that the straight forward method of transforming the formula into a polynomial as a sum of monoms might lead to exponentially many (intermediate) monoms: consider a threshold formula $F = \mathrm{Th}_k^n(x_1, \ldots, x_n)$ that evaluates to 1 if at least k of its n inputs are 1's. The corresponding polynomial p_F over GF(2) has the following form.

$$p_F \quad = \quad \sum_{i=k}^{n} \sum_{\substack{S \subseteq \{1,\ldots,n\} \\ \|S\|=i}} \prod_{s \in S} x_s \prod_{t \notin S} (x_t + 1).$$

Hence p_F consists of $\sum_{i=0}^{n-k} \binom{n}{i}$ monoms, each consisting of n terms of the form x_j or $1 + x_j$. We conclude that we can handle p_F efficiently if $n - k$ is constant. Since we are dealing with parity-over-threshold formulas, we can use the parity for negation. Hence, a similar argument as above works when the threshold k is bounded by a constant. In particular, we have an efficient algorithm if the fan-in of the threshold gate is bounded.

Proposition 2.5 *k- $\oplus$ -Th-SAT $\in$ **P**, for any $k > 0$.*

In the general case, the crucial observation is that we can evaluate the polynomial associated with a parity-over-threshold formula F without having a compact explicit way of writing it: F itself can be used to evaluate p_F by *dynamic programming*. Then we can continue as in the proof of Theorem 2.4.

Theorem 2.6 *$\oplus$-Th-SAT $\in$ **RP**.*

Proof. Consider the threshold formula $\mathrm{Th}_k^n(x_1, \ldots, x_n)$ that evaluates to 1 if at least k of its n inputs are 1's. Let $t_k^n(x_1, \ldots, x_n)$ be the polynomial over GF(2) that represents Th_k^n. We have the following recurrence relation for $1 \leq k < n$.

$$t_k^n(x_1, \ldots, x_n) \quad = \quad x_n \cdot t_{k-1}^{n-1}(x_1, \ldots, x_{n-1}) + (1 - x_n) \cdot t_k^{n-1}(x_1, \ldots, x_{n-1}). \tag{1}$$

The base cases are $t_l^l(x_1, \ldots, x_l) = x_1 \cdots x_l$ and $t_0^l(x_1, \ldots, x_l) = 1$, for $l = 1, \ldots, n$. Therefore, to evaluate t_k^n over GF(2) at a given point, we can use dynamic programming: we simply have to fill out a table with at most $n(k+1) = O(n^2)$ entries.

Now, let $F(x_1, \ldots x_n)$ be a parity-over-threshold formula having m threshold clauses. We can write the polynomial p_F representing F over GF(2) as follows.

$$p_F(x_1, \ldots, x_n) = \sum_{i=1}^{m} t_{k_i}^{n_i},$$

where each polynomial $t_{k_i}^{n_i}$ has some n_i literals as input. We can evaluate p_F over GF(2) at a given point by evaluating each of the $t_{k_i}^{n_i}$ as described above and then adding up all the values obtained. Now we can proceed as in Theorem 2.4. $\square$

Remark 2.7 $\oplus$- $\wedge$ -SAT *and* $\oplus$-Th-SAT *can be decided efficiently even* in parallel. *That is, we have in fact shown that these two problems are in* **RNC**. *For* $\oplus$- $\wedge$ -SAT *this is immediate from our algorithm (note that testing irreducibilty in small fields is in* $\mathbf{NC}^2$*). For* $\oplus$-Th-SAT , *there is a dynamic programming step which might not be parallelizable. However, in our case, the specific table we have to fill out according to equation (1) can be seen as a arithmetic circuit of degree* n *over* GF(2). *Miller, Ramachandran, and Kaltofen [MRK88] have shown that such circuits can be evaluated in* $\mathbf{NC}^2$.

Remark 2.8 *When we take modulo* m *gates instead of parity gates, all we need for the above algorithms to work correctly is that the formulas have some associated polynomial over some* field. *Therefore, we have* Mod(m)-Th-SAT $\in$ **RP** *for any* prime power m. *For* m *not being a prime power, the complexity of* Mod(m)- $\wedge$ -SAT *and* Mod(m)-Th-SAT *remains open.*

Finally, we show that Theorem 2.6 can be generalized to *weighted* threshold gates with polynomially bounded weights. This yields a randomized algorithm to solve the parity version of Integer Linear Programming in polynomial time.

Consider a weighted threshold gate

$$w_1 x_1 + \cdots + w_n x_n \quad \geq \quad k,$$

where $w_1, \ldots, w_n$ are integer weights. Let $wt_k^n(x_1, \ldots, x_n)$ be the polynomial over GF(2) that represents the weighted threshold gate. Observe that we still have a recurrence relation analogous to equation (1). Namely, let N be the sum of the negative weights and P be the sum of the positive weights. For $N < k < P$

$$wt_k^n(x_1, \ldots, x_n) \quad = \quad x_n \cdot wt_{k-w_n}^{n-1}(x_1, \ldots, x_{n-1}) + (1 - x_n) \cdot wt_k^{n-1}(x_1, \ldots, x_{n-1}).$$

The base cases are $wt_P^n(x_1, \ldots, x_n) = y_1 \cdots y_n$, where $y_i = x_i$ if $w_i > 0$, and $y_i = 1 - x_i$ otherwise, and $wt_N^n(x_1, \ldots, x_n) = 1$. Let $W = N + P = \|(w_1, \ldots, w_n)\|_1$. To evaluate t_k^n over GF(2) we have to fill out a table with at most $O(nW)$ entries. This can be done in polynomial time when W is polynomially bounded. It follows that $\oplus$-Th-SAT is in **RP** even with polynomially bounded threshold gates.

Now consider the *Integer Linear Programming problem, ILP*: given a $m \times n$ matrix A and a m vector b, is there a n vector x such that $Ax \geq b$, where A, b, and x are over the integers? ILP, and even 0-1-ILP, where x is required to be a zero-one vector, is **NP**-complete [Kar72]. For 0-1-ILP, note that each row in $Ax \geq b$ can be seen as a weighted threshold gate as above. By the preceeding discussion we conclude that the parity version of 0-1-ILP, call it odd-0-1-ILP, that asks for a zero-one vector x such that an odd number of the conditions is satisfied, is in **RP**.

Corollary 2.9 odd-0-1-ILP *with polynomially bounded weights is in* **RP**.

Clearly, Remark 2.7 and 2.8 apply to weighted $\oplus$-Th-SAT and odd-0-1-ILP as well.

3 Threshold at the Output

We start by considering $\wedge$-$\oplus$-SAT. Given a and-over-parity formula F, we are seeking for an assignment such that in *all* the parity clauses an odd number of literals has value one. This can be seen as a set of linear equations over GF(2) that have to be satisfied simultaneously. Therefore $\wedge$-$\oplus$-SAT can be solved in polynomial time, in fact in $\mathbf{NC}^2$.

Proposition 3.1 $\wedge$-$\oplus$-SAT $\in \mathbf{P}$.

Remark 3.2 *The approach via linear equations over* GF(2) *gives not only an efficient way to decide* $\wedge$-$\oplus$-SAT *but even to* count the number of solutions *in polynomial time. This contrasts with the fact that the counting version of* DNF-SAT *is* #**P**-*complete [Val79b].*

Next we generalize to threshold-over-parity formulas. In the above setting of linear equations, we ask, for some threshold value k, whether some k-subset of the equations is simultaneously satisfiable. In contrast to the case where all equations have to be fullfilled, there is no polynomial-time algorithm known for this generalization. We show below that the problem is in fact **NP**-complete. Before doing so, we argue that the problem is easy for certain threshold values k that depend on the number of clauses a formula has.

Consider a formula $F \in$ Th-$\oplus$-SAT and assume that no constants are fed into the threshold gate. We make a random assignment to the variables, that is, each variable is assigned independently a value 0 or 1 with equal probability. Then we expect a variable to have value 1 with probability 1/2. Also, for a clause $C = l_1 \oplus \cdots \oplus l_k$, we have $\mathbf{E}(C \text{ evaluates to } 1) = 1/2$. Because of the linearity of the expectation, a random assignment is expected to satisfy 1/2 of the clauses. Therefore there also exist such an assignment. We conlude that the problem is trivial for thresholds up to 1/2. However, Th-$\oplus$-SAT is **NP**-complete for thresholds (strictly) between 1/2 and 1.

Theorem 3.3 Th-$\oplus$-SAT *is* **NP**-*complete.*

Proof. We reduce 3-CNF-SAT to Th-$\oplus$-SAT. Consider a clause $C = l_1 \vee l_2 \vee l_3$ of a given 3-CNF formula F, for literals l_1, l_2, l_3. From C we define the following sequence of 7 formulas

$$\begin{array}{rcll} S(C) & = & (& l_1,\ l_2,\ l_3, \\ & & & l_1 \oplus l_2,\ l_1 \oplus l_3,\ l_2 \oplus l_3, \\ & & & l_1 \oplus l_2 \oplus l_3 \qquad\qquad\quad). \end{array}$$

Let a be an assignment to the literals. Observe that when a does *not* satisfy C, then none of the formulas of $S(C)$ is satisfied. On the other hand, if a satisfies C, then exactly four formulas of $S(C)$ are satisfied, no matter which of the literals of C are satisfied by a.

Let formula F have m clauses. We construct a threshold-over-parity formula G as follows: for each clause C of F, we put all formulas from $S(C)$ as input to the threshold-gate of G which therefore has fan-in $7m$. It follows that if an assignment a satisfies F, then exactly $4m$ inputs of the threshold-gate are on when a is given as input to G. If a does *not* satisfies F, then at most $4(m-1)$ inputs of the threshold-gate are on.

Thus, with a $4m$ threshold-gate on top, G is satisfiable if and only if F is satisfiable. In fact, by the above discussion, we could even use a $4m$ equality-gate. □

Remark 3.4 *If we use modulo m gates instead of parity gates, note first that* Th-Mod(m)-SAT *is clearly* **NP***-complete for $m > 3$, since in that case, the* modulo m*-gate behaves like an* or*-gate on clauses with 3 literals. So the only interesting case that remains is when $m = 3$: for a clause C consisting of the literals l_1, l_2, l_3, we consider the following sequence of 15 formulas.*

$$\begin{aligned} S(C) \;=\; (\;& l_1,\; l_2,\; l_3,\\ & l_1,\; l_2,\; l_3,\\ & 2l_1+l_2,\; 2l_1+l_3,\; 2l_2+l_3,\\ & 2l_1+l_2,\; 2l_1+l_3,\; 2l_2+l_3,\\ & 2l_1+l_2+l_3,\; l_1+2l_2+l_3,\; l_1+l_2+2l_3\;\;), \end{aligned}$$

where $+$ denotes addition in GF(3)*, i.e., according to the* modulo 3*-gate. Now, if clause C is not satisfied by an assignment, then none of the formulas of $S(C)$ have value* 1*. On the other hand, if an assignment satisfies C, then exactly 9 formulas from $S(C)$ have value* 1*. Then we can continue in the same way as in the proof of Theorem 3.3. It follows that* Th-Mod(3)-SAT *is* **NP***-complete.*

Note that this result also follows from the **NP***-completeness of* 2-MaxSAT *[GJS76], because again, a* modulo 3*-gate works like an* or*-gate on clauses with 2 literals.*

Finally, we consider the satisfiability problem when the or-gate of a DNF formula is replaced by a threshold-gate. Although DNF-SAT can be solved in logarithmic space, Th-$\wedge$-SAT is again a hard problem.

Theorem 3.5 Th-$\wedge$-SAT *is* **NP***-complete.*

Proof. We reduce 3-CNF-SAT to Th-$\wedge$-SAT. Consider a clause $C = l_1 \vee l_2 \vee l_3$ of a given 3-CNF formula F, for literals l_1, l_2, l_3. There are seven assignments that satisfy C: if assignment a satisfies C, then a satisfies exactly one of the following formula from the following sequence.

$$\begin{aligned} S(C) \;=\; (\;& l_1 \wedge l_2 \wedge l_3,\\ & \bar{l}_1 \wedge l_2 \wedge l_3,\; l_1 \wedge \bar{l}_2 \wedge l_3,\; l_1 \wedge l_2 \wedge \bar{l}_3,\\ & \bar{l}_1 \wedge \bar{l}_2 \wedge l_3,\; \bar{l}_1 \wedge l_2 \wedge \bar{l}_3,\; l_1 \wedge \bar{l}_2 \wedge \bar{l}_3\;\;). \end{aligned}$$

Let formula F have m clauses. To construct a threshold-over-And formula G, take a threshold-gate with fan-in $7m$. We put, for each clause C of F, the corresponding seven formulas $S(C)$ as inputs to the threshold-gate. It follows that if an assignment a satisfies F, then exactly m inputs of the threshold-gate are on when a is given as input to G. If a does *not* satisfies F, then at most $m-1$ inputs of the threshold-gate are on.

Thus, with a m threshold- (or equality-) gate on top, G is satisfiable if and only if F is satisfiable. □

4 Open Problems

What is the complexity of Mod(m)-$\wedge$-SAT and Mod(m)-Th-SAT, when m is not a prime power?

Acknowledgments

We want to thank V Vinay, who originated this research by asking for the complexity of $\oplus$- $\vee$ -SAT. Discussions with Lance Fortnow yielded Proposition 2.2 as a first step. He also helped in Theorem 2.6. Finally, we want to thank Marek Karpinski for very useful hints to the literature, and Jin-yi Cai for helpful discussions.

References

[Ber70] E. Berlekamp. Factoring Polynomials over Large Finite Fields. *Mathematics of Computation 24(111)*, 713-735, 1970.

[BCW80] M. Blum, A. Chandra, M. Wegman. Equivalence of free Boolean graphs can be decided probabilistically in polynomial time. *Information Processing Letters 10(2)*, 80-82, 1980.

[C95] N. Creignou. A Dichotomy Theorem for Maximum Generalized Satisfiabilty Problems. *Journal of Computer and System Sciences 51*, 511-522, 1995.

[CH96] N. Creignou and M. Hermann. Complexity of Generalized Satisfiabilty Counting Problems. *Information and Computation 125*, 1-12, 1996.

[Coo71] S. Cook. The Complexity of Theorem-Proving Procedures. In *3rd ACM Symposium on Theory of Computing (STOC)*, 151–158, 1971.

[EK89] A. Ehrenfeucht and M. Karpinski. The Computational Complexity of (XOR,AND)-Counting Problems. Technical Report at Universität Bonn, Germany, 1989.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[GJS76] M. Garey, D. Johnson, and L. Stockmeyer. Some Simplified NP-complete graph problems. *Theoretical Computer Science 1*, 237-267, 1976.

[GKS90] D. Grigoriev, M. Karpinski, and M. Singer. Fast Parallel Algorithms for Sparse Multivariate Polynomial Interpolation over Finite Fields. *SIAM Journal on Computing 19(6)*, 1059-1063, 1990.

[HU79] J. Hopcroft, J. Ullman. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 1979.

[IM83] O. Ibarra and S. Moran. Probabilistic Algorithms for Deciding Equivalence of Straight-Line Programs. *Journal of the ACM 30(1)*, 217-228, 1983.

[Kar72] R. Karp. Reducibility Among Combinatorial Problems. In R. Miller and J. Thatcher (eds.), *Complexity of Computer Computations*, 85-104, 1972, Plenum Press, New York.

[MRK88] G. Miller, V. Ramachandran, and E. Kaltofen. Efficient Parallel Evaluation of Straight-Line Code and Arithmetic Circuits. *SIAM Journal on Computing 17(4)*, 687-695, 1988.

[Pap94] C. Papadimitriou. *Computational Complexity*. Addison-Wesley 1994.

[Scha78] T. Schaefer. The Complexity of Satisfiability Problems. In *10th ACM Symposium on Theory of Computing (STOC)*, 216-226, 1978.

[Schw80] J. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *Journal of the ACM 27(4)*, 701-717, 1980.

[Val79a] L. Valiant. The Complexity of Computing the Permanent. *Theoretical Computer Science 8*, 189-201, 1979.

[Val79b] L. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM Journal on Computing 8(3)*, 410-421, 1979.

[vdWae70] B. van der Waerden. Algebra 1 und 2. Heidelberger Taschenbücher 1970.

[Zip79] R. Zippel. Probabilistic Algorithms for Sparse Polynomials. In *Proceedings of EUROSAM 79*, Springer Verlag, Lecture Notes in Computer Sience 72, 1979.