A Hierarchy of Tractable Subclasses for SAT and Counting SAT Problems

Ştefan Andrei
Lamar University
Department of Computer Science
Beaumont, USA
Stefan.Andrei@lamar.edu

Gheorghe Grigoraş
Cuza University of Iaşi
Department of Computer Science
Romania
grigoras@infoiasi.ro

Martin Rinard
Massachusetts Institute of Technology
Cambridge, USA
rinard@lcs.mit.edu

Roland Hock Chuan Yap
National University of Singapore
Singapore
ryap@comp.nus.edu.sg

Abstract

Finding subclasses of formulæ for which the SAT problem can be solved in polynomial time has been an important problem in computer science. We present a new hierarchy of propositional formulæ subclasses for which the SAT and counting SAT problems can be solved in polynomial time. Our tractable subclasses are those propositional formulæ in conjunctive normal form where any set of $k+1$ clauses are related, i.e., there exists at least one literal in one clause that appears negated in another clause of the considered set of $k+1$ clauses. We say this subclass of formulæ is of rank k and it is different from previously known subclasses that are solvable in polynomial time. This is an improvement over the SAT Dichotomy Theorem and the counting SAT Dichotomy Theorem, since our subclass can be moved out from the $\mathbb{NP}$-complete class to the $\mathbb{P}$ class. The membership problem for this new subclass can be solved in $O(n \cdot l^{k+1})$, where n, l and k are the number of variables, clauses and the rank ($1 \leq k \leq l-1$), respectively. We give an efficient algorithm to approximate the number of assignments for any arbitrary conjunctive normal form propositional formula by an upper bound.

1 Introduction

The SAT problem is one of the most central problems in computer science. It has applications in many areas of computer science, such as program verification using bounded model checking [22, 25] and bounded model construction [6,7], real-time and embedded systems verification [3,4,23], planning problem in artificial intelligence [3,19], and so on.

The first problem proved to be $\mathbb{NP}$-*complete* is due to Stephen Cook [10]. He showed that any decision problem $P \in \mathbb{NP}$ can be polynomially transformed to the SAT problem: "*Given a propositional formula F, is there a satisfying assignment for F?*". In the effort of separating the subclasses of formulæ for which there exist polynomial algorithms to solve the SAT problem from those for which it is unknown if there exist polynomial algorithms, Schaefer discovered the first result, called the *SAT Dichotomy Theorem* [26]. Dichotomy results in complexity theory are rare, in general. Some examples of Dichotomy theorems are the subgraph homeomorphism problems [14], the H-coloring of graphs [16], the counting SAT problem [11], and the propositional circumscription [21]. Efforts to improve some of these important dichotomy theorems have been done by researchers. For example, Istrate [17] described a version of Schaefer's Dichotomy Theorem including the subclass of CNF formulæ when each variable occurs at most twice.

Here is an equivalent re-statement of the original SAT Dichotomy Theorem [26], which separates the subclasses which belong to the $\mathbb{P}$ class from those belonging to the $\mathbb{NP}$-*complete* class:

The SAT Dichotomy Theorem. *If a class is defined by the set of all the subformulæ that are allowed in its formulæ, then the SAT problem can be solved in polynomial time for the classes of Horn formulæ, reverse Horn formulæ, $2CNF$ formulæ, $0-$ formulæ, $1-$valid formulæ, and affine formulæ. In all other cases, the SAT problem for the class is $\mathbb{NP}$-complete.*

In other words, the SAT Dichotomy Theorem states that the SAT problem, having as input a formula belonging to one of the above six subclasses, can be solved in polynomial time. For all the subclasses in the SAT Dichotomy Theorem, we shall present definitions and examples in Section 2. In this paper, we improve upon Schaefer's Dichotomy Theorem by describing new non-trivial and different subclasses of subformulæ (called $Rank_k$) for which the SAT problem can be solved in polynomial time, namely $\mathcal{O}(n \cdot l^{k+1})$, where n, l and k are the number of variables, clauses and the rank ($1 \leq k \leq l-1$), respectively. In this way, we partition the class of $\mathbb{NP}$-*complete* problems from the SAT Dichotomy Theorem by identifying the subclasses of rank k formulæ in the $\mathbb{P}$ class. It is useful to contrast our result with the SAT Dichotomy Theorem. In the SAT Dichotomy Theorem, all the classes of formulæ (and the subsequent one described later this section - #SAT Dichotomy Theorem) refer to instances where a set S of logical relations is initially given and then the clauses of the instances are allowed to contain relations obtained from those in S by arbitrary projections or identifications of variables. On the other hand, our subclass is defined differently and does not have this property. Instead, in our approach, each clause depends on the rest of the clauses. For example, $F_1 = (p \vee q) \wedge (\overline{q} \vee r)$, is a rank 1 formula (note that $\overline{q}$ means the negative literal of variable q), while $F_2 = (p \vee q) \wedge (q \vee r)$ is not a rank 1 formula, although both F_1 and F_2 contain the clause $(p \vee q)$.

In this paper, we prove an interesting result that identifies a new hierarchy of propositional formulæ subclasses (different than the above six subclasses from the SAT Dichotomy Theorem) for which not only the SAT problem, but also the counting SAT problem, can be solved in polynomial time. As stated in [15], counting problems are another type of interesting problems, but they might be intractable even if $\mathbb{P} = \mathbb{NP}$. In a counting problem P, the goal is to determine how many solutions exist, unlike in a decision problem where a simpler "Yes/No" answer suffices. The problem of counting the number of satisfying assignments (denoted by #SAT) was proved to be $\#\mathbb{P}-$complete [28]. The $\#\mathbb{P}-$complete problems are at least as hard as $\mathbb{NP}$-*complete* problems. The #SAT problem is a valuable approach for evaluating techniques in an effort to avoid computational difficulties, such as constraint satisfaction and knowledge compilation. Although the representative counting problem is the #SAT problem, counting has a major impact on many sub-areas of computer science. The SAT Dichotomy result has been extended to the #SAT problem by Creignou and Hermann in 1996. Their main result, Theorem 4.1 [11], is:

The Counting SAT Dichotomy Theorem. *If all logical relations used in generalized #SAT are affine, then the number of satisfying assignments can be computed in polynomial time, otherwise the problem is $\#\mathbb{P}-$complete.*

There have also been various efforts to determine the complexity of the #SAT problem for the polynomial solvable subclasses identified in Schaefer's Dichotomy Theorem [12, 17, 20].

Contribution of our paper: This paper describes new non-trivial and different subclasses of subformulæ for which the #SAT problem (and of course the SAT problem, too) can be solved in polynomial time. The first contribution of this paper is that the SAT and #SAT problems are tractable for our class of propositional formulæ. A particular instance of bounded rank formulæ, called *hitting formulæ*, was presented in [9]. The purpose of considering hitting formulæ was to investigate the closure under splitting. This subclass of formulæ coincides with our subclass of formulæ of rank 1. We present a simple example of our subclass of subformulæ which is not a member of any known subclasses from the SAT Dichotomy Theorem and the Counting SAT Dichotomy Theorem.

Example 1.1 *The propositional formula $F_3 = (p \vee q \vee r) \wedge (\overline{p} \vee \overline{q} \vee \overline{r})$ does not belong to any of subclasses from neither the SAT Dichotomy Theorem nor the counting SAT Dichotomy Theorem. Instead, it belongs to the bounded rank class of subformulæ (i.e., it has rank 1) as we shall see in Remark 3.2.* ■

The second contribution of the paper is a computationally efficient approximation for any arbitrary propositional formula. Briefly, for any clausal formula F, there exists at least a formula F' of bounded rank such that the number of satisfying assignments of F is less than the number of satisfying assignments of F' (Section 4). In constrast to this upper bound approximation of the number of satisfying assignments, paper [5] investigates efficient algorithms for estimating the lower bound approximation of the number of satisfying assignments.

The rest of this paper is organized as follows. Section 2 describes the necessary definitions, notations and related results. Section 3 defines our hierarchy of non-trivial bounded rank propositional formulæ for which the SAT and #SAT problems are tractable. Section 4 describes an efficient algorithm to determine a good upper bound for the number of satisfying assignments and Section 5 concludes this paper.

2 Preliminaries

We present some concepts and notations to allow the remainder of the paper to be self contained, by including some results and examples. In [18], an algorithm was introduced for testing satisfiability of a clausal formula F over n variables by counting, the satisfying assignments falsifying F using the inclusion-exclusion principle. A dual algorithm for deciding the satisfiability of a disjunctive normal formula was described in [27]. It may be regarded as a refinement of the Iwama's algorithm, and it is well behaved with a preconditioning on some parts of the Davis-Putnam's procedure. In [24], a similar algorithm is employed for counting satisfying assignments.

In [13], the formula for computing the number of solutions of a set of any clauses is established using the notion of independency between clauses. Furthermore, Dubois presented an efficient algorithm for counting satisfying assignments. In [29], an algorithm based on similar ideas and with similar time complexity like [13] was presented. In [1,2,5], the number of satisfying assignments was called the *determinant* of the clausal formula (because of the similarity with the determinant of the matrix for a linear algebraic system). Moreover, we presented an algorithm for satisfiability of a clausal formula based on the rules of pure literal, unitary resolution, primal bounded literal followed by comparison of its determinant to zero. Details about the complexity of many algorithms for counting satisfying assignments can be found in [8].

Let $\mathbb{LP}$ be the *propositional logic* over the finite set of *atomic formulae* (propositional variables) $V = \{A_1, ..., A_n\}$. A *literal* L is an atomic formula A (*positive* literal) or its negation $\neg A$ (*negative* literal), and denote $v(L) = v(\overline{L}) = A$. A function $S : V \rightarrow \{0, 1\}$ is a *satisfying assignment* and it can be uniquely extended in $\mathbb{LP}$ to a formula F. The binary vector $(y_1, ..., y_n)$ is a satisfying assignment for F over $V = \{A_1, ..., A_n\}$ if and only if $S(F) = 1$ such that $S(A_i) = y_i, \forall\ i \in \{1, ..., n\}$. Any propositional formulæ $F \in \mathbb{LP}$ can be translated into the *conjunctive normal form* (CNF): $F = (L_{1,1}\vee\ ...\ \vee L_{1,n_1})\wedge\ ...\ \wedge(L_{l,1}\vee\ ...\ \vee L_{l,n_l})$, where $L_{i,j}$ are literals and $l \geq 1$. We shall use the set representation $\{\{L_{1,1}, ..., L_{1,n_1}\}, ..., \{L_{l,1}, ..., L_{l,n_l}\}\}$ to denote F. Any finite disjunction of literals is a *clause* and a formula in CNF is called a *clausal formula.* The set of atomic formulae whose literals belong to clause C and formula F are denoted by $v(C)$ and $v(F)$, respectively. A formula F is called *tautology* if and only if for any structure S, it follows that $S(F) = 1$. A formula F is called *satisfiable* if and only if there exists a structure S for which $S(F) = 1$. A formula F is called *unsatisfiable* if and only if F is not satisfiable. In this paper, only *non-tautological* clauses (which have no simultaneous occurrences of a literal L and $\overline{L}$) are considered. We say that a clause C_1 is included in the clause C_2 or C_2 is a super-clause of C_1 (denoted by $C_1 \subseteq C_2$) if and only if $\forall\ L \in C_1$ we have $L \in C_2$. A finite non-tautological clause C constructed over V is *maximal* if and only if $v(C) = V$. A clausal formula is *maximal* if and only if it contains only maximal clauses. We denote by $\square$ the *empty clause* (i.e., the one without any literal) and by $\emptyset$ the empty set. A *unit clause* has only one literal. A clause C is called *Horn* (*reverse Horn*) if and only if C only one or zero positive (or negative) literal. A (reverse) Horn CNF formula has only (reverse) Horn clauses. A clause C is called *positive* (*negative*) if and only if C contains only positive (or negative) literals. A *positive* (*negative*) CNF formula has only positive (negative) clauses. Note that a negative (positive) formula is also a (reverse) Horn formula. A clause C is called *1-valid* (*0-valid*) if and only if C contains at least one positive (negative) literal. A *1-valid* (*0-valid*) CNF formula has only *1-valid* (*0-valid*) clauses. A clause C is called *2CNF* if and only if C contains at most two literals. A *2CNF* formula has only *2CNF* clauses. We will also make use of the class of afine formulæ. This class is special because it is usually expressed in a more compact linear equation form than in CNF. A *linear equation* has one of the forms $(x \oplus y \oplus z \oplus ... = 0)$ or $(x \oplus y \oplus z \oplus ... = 1)$, where $\oplus$ is the arithmetic addition modulo 2 and x, y, z ... are propositional variables. An *affine formula* is a conjunction of linear equations. For example, the affine formula $x \oplus y = 0$ corresponds to the CNF $\{\ \{x, y\}, \{\overline{x}, \overline{y}\}\ \}$, whereas $x \oplus y = 1$ corresponds to the CNF $\{\ \{\overline{x}, y\}, \{x, \overline{y}\}\ \}$. Moreover, it is known from linear algebra that the number of solutions of such conjunctions of linear equations is a power of 2. This implies that the number of satisfying assignments of an affine formula [26] is a power of 2.

For a finite set A, $|A|$ denotes the number of elements of A. The notations $\mathbb{Z}$, $\mathbb{N}$ and $\mathbb{N}_+$ stand the set of integers, positive integers, and the set of strict positive integers, respectively. The number of all sets with i elements from a set with n elements is denoted by $\binom{n}{i}$, and it is equal to $\frac{n!}{(n-i)!\cdot i!}$, where $n! = 1\cdot 2\cdot\ ...\ \cdot n$. for $n \geq 1$ and $0! = 1$.

We define $m_V(S)$ as $|\ \{v \in V :$ neither v nor $\neg v$ appears in any clause of $S\}\ |$. In other words, $m_V(S)$ denotes the number of variables from V which do not occur in $v(S)$. Obviously, $0 \leq m_V(S) \leq |V|$. The boundary cases are obtained when $v(S) = V$ or S is the empty clause. Given S the set of clauses $\{C_1, ..., C_l\}$, we say that S is reducible if and only if there exist two clauses C_i and C_j, where $i \neq j$, such that there exists a literal $L \in C_i$ and $\neg L \in C_j$. We say that the set of clauses S is irreducible if and only if S is not reducible. Then we define $dif_V(S) = 0$ if S is reducible and $2^{m_V(S)}$ if S is irreducible. Since the empty set is irreducible, then $dif_V(\square) = 2^{|V|}$. Finally we define

$$det_V(S) = \sum_{R\subseteq S}(-1)^{|R|}dif_V(R)$$

and call it the *determinant* of the set S of clauses. The notion of determinant appeared in [1, 2, 4, 5], but this revised definition above is more simplified and much clearer. Moreover it better highlights that only irreducible clauses are contributing to the determinant computation.

Example 2.1 *Let* $F_4 = \{C_1, C_2, C_3\}$*, where* $V = \{p, q, r\}$ *and clauses* $C_1 = \{p, \overline{q}\}$, $C_2 = \{q, \overline{r}\}$, $C_3 = \{p, r\}$*. Then* $m_V(C_1) = 1$, $m_V(C_1, C_2) = 0$, *and so on. Thus,* $dif_V(C_1, C_3) = 1$, $dif_V(C_1, C_2, C_3) = 0$, *etc. Therefore,* $det_V(F_4) = 2^3 - (2^1 + 2^1 + 2^1) + 2^0 = 3$. ■

The next result establishes the link between the determinant of a clausal formula and its satisfiability (proved in [2]). In other words, the determinant of a clausal formula coincides with the number of satisfying assignments of that formula.

Theorem 2.1 *Let $F \in \mathbb{LP}$ over V. Then there exist $det_V(F)$ number of satisfying assignments for F.*

According to Theorem 2.1, it follows formula F given in Example 2.1 is satisfiable and has $det_V(F) = 3$ satisfying assignments.

Definition 2.1 *Given $F = \{C_1, ..., C_l\}$ a clausal formula over V, then F has rank k if $dif_V(C_{i_1}, ..., C_{i_{k+1}}) = 0$, for any $i_1, ..., i_{k+1}$ distinct indexes from $\{1, ..., l\}$. We denote by $Rank_{V,l,k}$ the set of all propositional formulæ over V, and having l clauses and rank k. When the context is clear, the indexes V and l may be omitted for simplicity.*

According to the above definition, it can be easily seen that $F_1 = \{ \{p, q\}, \{\overline{q}, r\}\}$ over $V = \{p, q, r\}$ is a rank 1 formula, the reason being $dif_V(\{p, q\}, \{\overline{q}, r\}) = 0$. The formula F_4 from Example 2.1 is not a rank 1 formula because $dif_V(C_1, C_3) \neq 0$. Since $dif_V(C_1, C_2, C_3) = 0$, it follows instead that $F_4 \in Rank_2$.

3 Counting for bounded-rank formulæ

This section is devoted to introducing a new subclass of formulæ for which there exists a polynomial algorithm for solving both SAT and #SAT problems. We shall also make a comparison with other known subclasses of formulæ for which the SAT problem is tractable.

According to rank's definition, it follows that the membership problem ("Is F a $Rank_k$ formula?") can be solved in polynomial time. Note that k above is a given constant. That is, given a related formula $F = \{C_1, ..., C_l\}$ over $V = \{A_1, ..., A_n\}$, it can be checked in $O(n \cdot l^{k+1})$ whether F is a $Rank_k$ formula. The next result proves that our subclasses of rank k formulæ represent a hierarchy, that is, $Rank_1 \subseteq Rank_2 \subseteq ... \subseteq Rank_{l-1}$.

Lemma 3.1 *For every $k \geq 2$, the class of rank k formulæ contains the class of rank $(k-1)$ formulæ.*

Proof Let $F = \{C_1, ..., C_l\}$ be a propositional formula of rank $(k-1)$. Then $dif_V(C_{i_1}, ..., C_{i_k}) = 0$, for any $i_1, ..., i_k$ distinct indexes from $\{1, ..., l\}$. Obviously, for any $i_k \in \{1, ..., l\}$, we get $dif_V(C_{i_1}, ..., C_{i_k}, C_{i_{k+1}}) = 0$. This implies that F has rank k, too. ■

Lemma 3.1 shows that the higher rank a subclass has, the fewer positive and negative occurrences a literal has. Since this property relates to reducibility, we remind that a formula is irreducible if any literal appears either positive or negative in all clauses. Let us denote by $Irreducible_{V,l}$ the class of all irreducible formulæ over V having l clauses. For example $F_5 = \{ \{p, \overline{q}\}, \{p, r\}, \{\overline{q}, r\}\}$ over $V = \{p, q, r\}$ is an irreducible formula since all the literals appear either positive or negative. By doing a proper substitution, any irreducible formula can be converted to a positive (or negative) formula. For example, by substituting q into $\overline{q}$, F_5 from above becomes the following positive formula $F_6 = \{ \{p, q\}, \{p, r\}, \{q, r\}\}$. Clearly, any positive (or negative) formula is satisfiable (e.g., $S(A) = 1$ for any variable A, is a proper assignment for a given positive formula). The following remark shows that the set $Rank_{l-1}$ contains all propositional formulæ, except the irreducible ones.

Remark 3.1 *Let us denote by $\mathcal{CNF}_{V,l}$ the class of all clausal formulæ over V having l clauses. The following facts hold:*

1. *$\mathcal{CNF}_{V,l} = Rank_{V,l,l-1} \cup Irreducible_{V,l}$;*
2. *$Rank_{V,l,l-1} \cap Irreducible_{V,l} = \emptyset$.*

For the first item, it suffices to check that any clausal formula is either a rank $l-1$ or an irreducible formula. Let $F = \{C_1, ..., C_l\}$ be a clausal formula over V. We distinguish two cases:

i) there exists a literal L such that $L, \overline{L} \in C_1 \cup ... \cup C_l$. Then according to the rank's definition, F is a rank $l-1$ formula;

ii) otherwise, all literals appear either positive or negative in the clauses. This means that F is an irreducible formula.

For the second item, it is obvious that a rank $l-1$ formula cannot be an irreducible formula. Moreover, whenever $F = \{C_1, ..., C_l\}$ is an irreducible formula, then $dif_V(C_1, ... C_l) \neq 0$, and hence F cannot be a rank $l-1$ formula.

Lemma 3.1 and Remark 3.1 can be illustrated in Figure 1. By doing the union of the $Irreducible_{V,l}$ to the rank $l-1$ class of formulæ, the whole set of formulæ having l clauses is obtained.

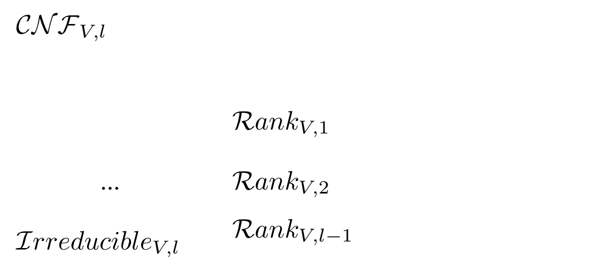

Figure 1. The set of all propositional formulæ

The determinant for the rank k formulæ can be eficiently computed as described in the next result.

Theorem 3.1 *The SAT and #SAT problems are tractable for CNF formulas of bounded rank.*

Proof Let $F = \{C_1, ..., C_l\}$ be a propositional formula of rank k, where $k \in \mathbb{N}_+$. Firstly, we shall prove by induction on $p \in \{k, ..., l\}$ that $dif_V(C_{i_1}, ..., C_{i_p}) = 0$. The base ($p = 2$) holds due to rank's definition. To prove the inductive step, let us suppose that $dif_V(C_{i_1}, ..., C_{i_m}) = 0$, where $m \in \{k, ..., l\}$. By applying the definition of dif_V, it implies that there exists a literal L such that $L \in C_{i_s}$ and $\overline{L} \in C_{i_t}$, where s and t are from $\{1, ..., m\}$. So, it follows immediately that $dif_V(C_{i_1}, ..., C_{i_m}, C_{i_{m+1}}) = 0$, where $i_{m+1} \in \{k, ..., l\}$.

Secondly, by applying the assertions $dif_V(C_{i_1}, ..., C_{i_p}) = 0$, for all $p \in \{k, ..., l\}$, in the definition of the determinant, it follows that $det_V(F) = \sum_{R\subseteq F, |R|\leq k} (-1)^{|R|} \cdot dif_V(R)$. Thus the determinant of any rank k formula $F = \{C_1, ..., C_l\}$ over $V = \{A_1, ..., A_n\}$ can be computed in $\mathcal{O}(n \cdot l^{k+1})$. Therefore, the #SAT problem for the rank k formulæ, where k is fixed, is tractable. This implies the SAT problem is tractable, too. ■

We now show that the rank 1 formulæ are disjoint from the existing classes in the SAT and counting SAT Dichotomy Theorems.

Remark 3.2 *The class of $Rank_1$ formulæ is different than the classes of Horn, inverse Horn, Krom, $0-$valid, $1-$valid, and affine formulæ. Consider $F_7 = \{\{p, \overline{q}\}, \{\overline{p}, q\}, \{q, \overline{r}\}, \{\overline{q}, r\}\}$, over $V = \{p, q, r\}$. It can immediately be seen that F_7 is a Horn, inverse Horn, Krom, $0-$valid, $1-$valid, and affine formula. It is affine because F_7 is equivalent to the system of linear equations: $p \oplus q = 1 \wedge q \oplus r = 1$. On the other hand, F_7 is not a rank 1 formula because $dif_V(\{p, \overline{q}\}, \{\overline{q}, r\}) \neq 0$).*

To disprove the other inclusion, let us consider $F_3 = \{\{p, q, r\}, \{\overline{p}, \overline{q}, \overline{r}\}\}$ over V (that is, the formula from Example 1.1). It can be checked that F_3 is not a Horn, inverse Horn, Krom, $0-$valid, $1-$valid, or affine formula. The reason that F_3 is not an affine formula is due to the fact that $det_V(F_3) = 6$, which is not a power of 2. Instead F_3 is a rank 1 formula because $dif_V(\{p, q, r\}, \{\overline{p}, \overline{q}, \overline{r}\}) = 0$.

An interesting question related to Figure **1** is how large is the class of rank k formulæ, for $1 \leq k \leq l-1$? The next result proves that there exist a double exponential number of rank k formulæ over an alphabet with n variables, e.g., a sum of terms that contains 2^{2^n}. In other words, the class $Rank_k$ is a non-trivial new subclass of formulæ for which both the SAT and counting SAT problems are tractable.

Theorem 3.2 *Given $n = |V|$ and $k \geq 2$, the following facts hold:*

1) *there exist $2^{3^n} - 2^{n+2^n}$ different rank $l-1$ formulæ;*

2) *there exist at least $\binom{n}{n} \cdot 2^{2^n} + \binom{n}{n-1} \cdot 2^{2^{n-1}} + ... + \binom{n}{1} \cdot 2^{2^1} + \binom{n}{0} \cdot 2^{2^0} - 3^n + 2^n - 2n - 4$ different rank k formulæ over V.*

Proof 1) According to Remark 3.1, the number of rank $l-1$ formulæ is the difference between the total number of non-tautological clausal formulæ and the total number of irreducible formulæ.

Given F, any literal may occur positive, negative or not at all. Then, the total number of clauses which can be build up with all the n variables equals to the number of total functions from $\{1, 2, ..., n\}$ to $\{-1, 0, 1\}$, which is 3^n. Certainly, we have to use the set representation for formulae (thus, eliminating multiple occurences of literals and clauses). Moreover, the clauses containing both (a literal) L and $\overline{L}$ have to be (syntactically) eliminated. Since we need to count all subformulæ of the formula containing all the possible clauses, it means that the number of the whole set of clausal formulæ is 2^{3^n}.

To count the number of irreducible formulæ, we count first the total number of positive clauses that can be constructed with all the n variables. This coincides the number of total functions from $\{1, ..., n\}$ to $\{0, 1\}$, that is, 2^n. An irreducible formula can be obtained by substituting any positive literal with its negation. The number of all these substitutions equals the number of total functions from $\{1, ..., n\}$ to $\{-1, 1\}$, that is, 2^n. Therefore, the number of irreducible formulæ is $2^{2^n} \cdot 2^n = 2^{n+2^n}$.

2) We highlight a double exponential number of rank 1 formulæ. Hence, there will be at least as many this number of rank k formulæ, where $k \in \mathbb{N}_+$. Let us note that once we identify a rank 1 formula F, then every subformula having at least two clauses of F is a rank 1 formula, too. Since clausal formulæ are represented as sets, given $F = \{C_1, ..., C_l\}$ a rank 1 formula, there exist $2^l - l - 1$ different rank 1 formulæ. Suppose that the alphabet is $V = \{A_1, ..., A_n\}$. In order to count a subformula only once, we shall find some formulæ and then generate all its subformulæ as explained earlier. For example, one such rank 1 formula is the one having all the maximal clauses. Obviously, this formula has 2^n clauses. This means there exist $2^{2^n} - 2^n - 1$ different rank 1 subformulæ having only maximal clauses. Similarly, we can find all the rank 1 formulæ for which all their clauses have exactly $n-1$ literals. This number is $\binom{n}{n-1} \cdot (2^{2^{n-1}} - 2^{n-1} - 1)$. Continuing this generation process until unit clauses are considered, we get $\binom{n}{1} \cdot (2^{2^1} - 2^1 - 1)$ different rank 1 formulæ containing only unit clauses.

We can go further and identify other rank 1 subformulæ which were not generated in the previous generation process. In order to avoid generation of the same rank 1 subformulæ, we consider $F_8 = \{\{A_1, \overline{A_2}, ..., \overline{A_n}\}, \{A_2, \overline{A_3}, ..., \overline{A_n}\}, ..., \{A_{n-1}, \overline{A_n}\}, \{A_n\}\}$ and its "reverse" $F_9 = \{\{\overline{A_1}, A_2, ..., A_n\}, \{\overline{A_2}, A_3, ..., A_n\}, ..., \{\overline{A_{n-1}}, A_n\}, \{\overline{A_n}\}\}$.

Obviously, F_8 and F_9 are rank 1 formulæ. They can generate $2 \cdot (2^n - n - 1)$ different rank 1 formulæ. Counting all the above mentioned rank 1 subformulæ, there exist at least $\binom{n}{n} \cdot 2^{2^n} + \binom{n}{n-1} \cdot 2^{2^{n-1}} + ... + \binom{n}{1} \cdot 2^{2^1} + \binom{n}{0} \cdot 2^{2^0} -$

$3^n + 2^n - 2n - 4$ different rank 1 subformulæ. ■

This section concludes that our subclasses of rank k formulæ represent a large and non-trivial class of formulæ among the total number of all clausal formulæ (that is, 2^{3^n}). The SAT and #SAT problems are tractable for rank k formulæ. These subclasses are different than those classes for which the considered problems are known to be in the $\mathbb{P}$ class.

4 A Least Upper Approximation

This section is devoted to a polynomial time approximation of the determinant for an arbitrary clausal formula (i.e., not necessarily a rank k formula). Briefly, we shall prove that for any clausal formula F over V, there exists a (minimum) rank k formula F' over V such that $det_V(F) \leq det_V(F')$.

Given two arbitrary clausal formulæ $F = \{C_1, ..., C_l\}$ and $F' = \{C'_1, ..., C'_l\}$ over V, we define the *internal order*, as $F \subseteq_{in} F'$ if and only if $\forall\ C_i \in F$, $i \in \{1, ..., l\}$, there exists C'_i such that $C_i \subseteq C'_i$. Note that in this definition, F and F' have the same number of clauses and the latter $\subseteq$ simply means the ordinary set inclusion.

Next, a result which allows shrinking the clauses of a given clausal formula is presented. This result helps finding an approximation of the determinant for any arbitrary clausal formula.

Lemma 4.1 *Let F and F' be two arbitrary clausal formulæ over V such that $F \subseteq_{in} F'$. Then $det_V(F) \leq det_V(F')$.*

Proof By definition, every assignment satisfying C_i also satisfies any C'_i that contains C_i. Since $F \subseteq_{in} F'$, every assignment satisfying F also satisfies F'. By Theorem 2.1, it follows that $det_V(F) \leq det_V(F')$. ■

Definition 4.1 *Given $F = \{C_1, ..., C_l\}$ a clausal formula over V, we call $UA_k(F)$ the set of* upper approximations *of F, as the set of all rank k formulæ obtained from F by adding some new literals to its clauses. In order to have an efficient approximation, we define the set of* least upper approximations *of F, as $LUA_k(F) = \{F' \mid F' \in UA_k(F)$, there is no $F'' \in UA_k(F)$ such that $F'' \subset_{in} F'\}$.*

The notation $F'' \subset_{in} F'$ means $F'' \subseteq_{in} F'$ and $F'' \neq F'$. In other words, $LUA_k(F)$ represents the set of all rank k formulæ obtained from F by adding a minimum number of literals until they enjoy the bounded rank property.

Example 4.1 *Consider $F_{10} = \{\{p\}, \{q, \overline{r}\}\}$ a clausal formula over $V = \{p, q, r\}$. Then, the set of least upper approximations of F_{10} is $LUA_2(F) = \{\ F_{11}, F_{12}, F_{13}\ \}$, where $F_{11} = \{\{p, \overline{q}\}, \{q, \overline{r}\}\}$, $F_{12} = \{\{p, r\}, \{q, \overline{r}\}\}$, and $F_{13} = \{\{p\}, \{\overline{p}, q, \overline{r}\}\}$. For example, $\{\{p, \overline{q}, r\}, \{q, \overline{r}\}\}$ is in $UA_2(F_{10}) - LUA_2(F_{10})$.* ■

The next proposition represents an approximation result for an arbitrary clausal formula.

Proposition 4.1 *Let F be a clausal formula over V. Then $det_V(F) \leq det_V(F'), \forall\ F' \in UA(F)$.*

Proof According to definition of $UA_k(F)$, it follows that F' has clauses with more literals than clauses of F, or equivalently $F \subseteq_{in} F'$. By Lemma 4.1, we get $det_V(F) \leq det_V(F')$. ■

Example 4.1 demonstrates that there might be many rank k formulæ associated to a given clausal formula F. It is desirable to identify $F' \in UA_k(F)$ having a small determinant. According to the determinant monotony property (Lemma 4.1), any $F' \in LUA_k(F)$ is a possible formula having a smaller determinant than F. However, this condition is not sufficient to get a minimum determinant. To demonstrate this, consider F_{10} from Example 4.1, for which we know that $det_V(F_{10}) = 3$. All three F_{11}, F_{12} and F_{13} are least upper approximations, but $det_V(F_{11}) = 4$, $det_V(F_{12}) = 4$ and $det_V(F_{13}) = 3$. So, even if we add one single literal for F_{11}, F_{12} and F_{13}, it seems that F_{13} is a better choice since it has the smallest determinant. Actually $det_V(F_{13}) = det_V(F_{10})$, which means F_{13} is actually the best least upper approximations of F_{10}.

4.1 Finding Good Upper Approximations

We will focus only on the rank 1 class of formulæ because this corresponds to the lowest time-complexity among the hierarchy of rank k formulæ. In order to find computationally efficient good upper approximation, we define the *left* and *right extension* for clauses. Here, a good approximation refers to one that is able to achieve the following:

1. the approximation is obtained by an efficient algorithm;

2. it should provide a small determinant.

Let us consider C_i, C_j two arbitrary clauses such that $i < j$ and $dif_V(C_i, C_j) \neq 0$. By *left extension*, we refer to the case $C_j \not\subseteq C_i$. The case $C_j \subseteq C_i$ can be solved immediately by removing C_i. The reason is because $det_V(F) = det_V(F - C_i)$. Here it is a sketch of the proof. Assuming $C_j \subseteq C_i$, then $dif_V(C_j, C_i, C_{i_1}, ..., C_{i_s}) = dif_V(C_i, C_{i_1}, ..., C_{i_s})$. In the definition of $det_V(F)$, the terms $dif_V(C_j, C_i, C_{i_1}, ..., C_{i_s})$ appear with a contrary sign to $dif_V(C_i, C_{i_1}, ..., C_{i_s})$, so all the terms containing C_i will disappear. Therefore $det_V(F) = det_V(F - C_i)$. Since $C_j \not\subseteq C_i$, we get that there exists a literal L from C_j which does not occur in C_i. Then the clause $C'_i = C_i \cup \{\overline{L}\}$ is an approximation of C_i. Hence $dif_V(C'_i, C_j) = 0$.

Similarly, we define the *right extension*, by referring to the case when $C_i \not\subseteq C_j$. It follows that there exists a literal L from C_i which does not occur in C_j. Then the clause

$C'_j = C_j \cup \{\overline{L}\}$ is an approximation of C_j. Therefore $dif_V(C_i, C'_j) = 0$.

If two clauses cannot be extended neither by left extension nor by right extension, then they coincide (so one of them can be removed from the set of initial clauses). Applying left and right extensions directly, the technique may not really provide the best approximation. For instance, considering F_{10} from Example 4.1, we get F_{11} as the approximation, which is not the best one.

We prove an auxiliary result that allows getting a better approximation.

Lemma 4.2 *Let C_i and C_j be two clauses over V, satisfying the properties: $\exists L_{j_1} \in C_j$ such that $L_{j_1}, \overline{L_{j_1}} \notin C_i$ and $\exists L_{i_1} \in C_i$ such that $L_{i_1}, \overline{L_{i_1}} \notin C_j$. Let us denote with u the number of literals of C_i which does not appear in C_j and with t the number of literals of C_j which does not appear in C_i. If $u \geq t$ then $dif_V(C_i \cup \{\overline{L_{j_1}}\})+ dif_V(C_j) \geq dif_V(C_j \cup \{\overline{L_{i_1}}\})+ dif_V(C_i)$.*

Proof Let us denote $C_i = \{L_{c_1}, ..., L_{c_s}, L_{i_1}, ..., L_{i_u}\}$ and $C_j = \{L_{c_1}, ..., L_{c_s}, L_{j_1}, ..., L_{j_t}\}$, where $L_{i_1}, ..., L_{i_u} \notin C_j$ and $L_{j_1}, ..., L_{j_t} \notin C_i$. We know that either C_i or C_j is modified.

In the first case (when C_i is modified), we get $dif_V(C_i \cup \{\overline{L_{j_1}}\}+ dif_V(C_j) = 2^{|V|-s-u-1}+ 2^{|V|-s-t}$. In the second case (when C_j is modified), we get $dif_V(C_j \cup \{\overline{L_{i_1}}\}+ dif_V(C_i) = 2^{|V|-s-t-1}+ 2^{|V|-s-u}$. The conclusion holds since the inequality $2^{|V|-s-u-1}+ 2^{|V|-s-t} \geq 2^{|V|-s-t-1}+ 2^{|V|-s-u}$ is equivalent to $u \geq t$. ■

Consider now the general case, that is, $F = \{C_1, ..., C_l\}$, where $l \geq 2$. We now present Algorithm **A** which takes into account Lemma 4.2 and generates in the output $F' = \{C'_1, ..., C'_l\}$, that is, a rank 1 approximation for F using left and right extension techniques.

Algorithm A:

Input: $F = \{C_1, ..., C_l\}$ a propositional formula, where $l \geq 2$;

Output: $F' = \{C'_1, ..., C'_l\}$ a rank 1 propositional formula such that $det_V(F) \leq det_V(F')$;

Method:

```
1. $F' = F$;
2. for $(i = 1; i < l; i++)$
3.  for $(j = i + 1; j <= l; j++)$
4.  if $(dif_V(C_i, C_j) > 0)$ {
5.     let $u$ be the number of literals of $C_i$ which do not appear in $C_j$
6.     let $t$ be the number of literals of $C_j$ which do not appear in $C_i$
7.     if $(u \geq t)$ {
8.        if $(\exists L \in C_j, L, \overline{L} \notin C_i)$ $C_i = C_i \cup \{\overline{L}\}$; // left extension
          else
9.        if $(\exists L \in C_i, L, \overline{L} \notin C_j)$ $C_j = C_j \cup \{\overline{L}\}$; // right extension
             else remove $C_j$
          else
10.         if $(\exists L \in C_i, L, \overline{L} \notin C_j)$ $C_j = C_j \cup \{\overline{L}\}$; // right extension
          else
11.         if $(\exists L \in C_j, L, \overline{L} \notin C_i)$ $C_i = C_i \cup \{\overline{L}\}$; // left extension
      }
```

By applying Algorithm **A** to F_{10} from Example 4.1, we get F_{13} that represents a better approximation than F_{11}. The next result proves the correctness and time complexity of Algorithm **A**.

Theorem 4.1 *Given $F = \{C_1, ..., C_l\}$ a propositional formula over V, where $|V| = n$, Algorithm **A** will provide a rank 1 approximation of F. Moreover, Algorithm **A** has an $\mathcal{O}(n \cdot l^2)$ time complexity.*

Proof Lemma 4.2 shows that the test from line **7** of Algorithm **A** will give a better approximation. That is, if $u < t$ then the lines **11** and **10** will be taken into consideration instead of lines **8** and **9**, respectively. The supplementary test $u < t$ can be done in constant time. Since C_i and C_j do not have any complimentary literals, $u = |C_i| - |C_i \cap C_j|$. Similarly, $t = |C_j| - |C_i \cap C_j|$. It follows that $u \geq t$ if and only if $|C_i| \geq |C_j|$, which can be tested in constant time. Hence Algorithm **A** has $O(n \cdot l^2)$ time complexity. ■

Algorithm **A** is very efficient because it does not increase the time complexity of the membership problem. According to Theorem 3.1, the membership problem for rank 1 formulæ has a time complexity of $\mathcal{O}(n \cdot l^2)$.

5 Conclusions

In this paper, we have introduced new subclasses of clausal formulæ for which there exist polynomial algorithms to solve the SAT problem and #SAT problem, too. For any arbitrary propositional formula, our approach provides an upper bound of its determinant.

References

[1] S. Andrei. The determinant of the boolean formulae. *Scientific Annals of the Bucharest University, Computer Science Section*, XLIV:83–92, 1995.

[2] S. Andrei. Counting for satisfiability by inverting resolution. *Artificial Intelligence Review*, 22(4):339–366, 2004.

[3] S. Andrei and A. Cheng. Verifying linear real-time logic specifications. In *RTSS07: Proceedings of the 28th IEEE Real-Time Systems Symposium*, pages 333–342, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[4] S. Andrei and W.-N. Chin. Incremental satisfiability counting for real-time systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 482–489, 2004.

[5] S. Andrei, G. Manolache, R. H. C. Yap, and V. Felea. Approximate satisfiability counting. In *SYNASC '07: Proceedings of the Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 196–202, Washington, DC, USA, 2007. IEEE Computer Society.

[6] A. Ayari and D. A. Basin. Bounded model construction for monadic second-order logics. In *CAV '00: Proceedings of the 12th International Conference on Computer Aided Verification*, pages 99–112, London, UK, 2000. Springer-Verlag.

[7] A. Ayari and D. A. Basin. QUBOS: Deciding quantified boolean logic using propositional satisfiability solvers. In *FMCAD '02: Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design*, pages 187–201, London, UK, 2002. Springer-Verlag.

[8] E. Birnbaum and E. L. Lozinskii. The good old Davis-Putnam procedure helps counting models. *J. Artif. Intell. Res. (JAIR)*, 10:457–477, 1999.

[9] H. K. Büning and X. Zhao. On the structure of some classes of minimal unsatisfiable formulas. *Discrete Applied Mathematics*, 130(2):185–207, 2003.

[10] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. Third Annual ACM Symposium Theory of Computing*, pages 151–158, 1971.

[11] N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 125(1):1–12, 1996.

[12] A. Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001.

[13] O. Dubois. Counting the number of solutions for instances of satisfiability. *Theor. Comput. Sci.*, 81(1):49–64, 1991.

[14] S. Fortune, J. E. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10:111–121, 1980.

[15] M. R. Garey and D. S. Johnson. *Computer and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[16] P. Hell and N. Nesetril. On the complexity of h-coloring. *J. Combin. Theory Ser. B*, 48:92–110, 1990.

[17] G. Istrate. Counting, structure identification and maximum consistency for binary constraint satisfaction problems. In *CP, LNCS 1330*, pages 136–149, 1997.

[18] K. Iwama. CNF satisfiability test by counting and polynomial average time. *SIAM J. Comput.*, 18(2):385–391, 1989.

[19] H. Kautz and B. Selman. Encoding domain knowledge for propositional planning. *Logic-based artificial intelligence*, pages 170–209, 2001.

[20] D. J. Kavvadias, M. Sideri, and E. C. Stavropoulos. Generating all maximal models of a boolean expression. *Inf. Process. Lett.*, 74(3-4):157–162, 2000.

[21] L. M. Kirousis and P. G. Kolaitis. A dichotomy in the complexity of propositional circumscription. *Theory Comput. Syst.*, 37(6):695–715, 2004.

[22] S. Krishnamurthi and K. Fisler. Foundations of incremental aspect model-checking. *ACM Trans. Softw. Eng. Methodol.*, 16(2):7, 2007.

[23] A. Lomuscio, W. Penczek, and B. Woźna. Bounded model checking for knowledge and real time. *Artif. Intell.*, 171(16-17):1011–1038, 2007.

[24] E. L. Lozinskii. Counting propositional models. *Inf. Process. Lett.*, 41(6):327–332, 1992.

[25] J. Marques-Silva. Interpolant learning and reuse in sat-based model checking. *Electron. Notes Theor. Comput. Sci.*, 174(3):31–43, 2007.

[26] T. J. Schaefer. The complexity of satisfiability problems. In *STOC*, pages 216–226, 1978.

[27] Y. Tanaka. A dual algorithm for the satisfiability problem. *Inf. Process. Lett.*, 37(2):85–89, 1991.

[28] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

[29] W. Zhang. Number of models and satisfiability of sets of clauses. *Theor. Comput. Sci.*, 155(1):277–288, 1996.