

The Representational Power of Conjunctive Normal Form

Thomas E. O'Neil

Computer Science Department
University of North Dakota
Grand Forks, ND 58202-9015

Jason J. Smith

Mathematics Department
University of North Dakota
Grand Forks, ND 58202-8376

Abstract - *There is continuing research interest in comparison of the complexity of problems within the class NP-Complete. This paper examines the representational power of conjunctive normal form Boolean expressions to establish a proper hierarchy for finite languages, where the language of an expression is defined to be the set of bit strings corresponding to its satisfying assignments. The increasing representational complexity parallels and perhaps explains the increasing complexity of k -SAT algorithms for higher values of k . The hierarchy provides a general framework for comparisons of satisfiability algorithms and leads to the conclusion that logical resolution is not a good algorithm for satisfiability testing.*

Keywords: satisfiability, conjunctive normal form, NP-complete, resolution, complexity of finite sets.

1 Overview

k -SAT is the problem of determining the satisfiability of a Boolean expression where the expression is in conjunctive normal form with at most k literals per clause (k -CNF). A CNF expression is a conjunction of clauses where each clause is a disjunction of literals, and each literal is a variable or a negated variable. k -SAT is a well-known NP-complete problem [1] for $k > 2$. It is used as a benchmark for the class NP-complete, since the critical region of the problem space where instances are most difficult is easily identified, and random expression generators can reliably produce problems in this region by adjusting k and the ratio of the number of clauses in an expression to the number of variables. All known algorithms require exponential time for critical region problems, both in the worst case and in the average case. In addition, both the worst-case and the average-case time complexity becomes increasingly exponential approaching a limit of 2^n as k increases beyond 3. Worst-case analysis of Monien and Speckenmeyer's branching algorithm [8], for example, classifies the problem as $O(1.62^n)$ for $k = 3$, $O(1.84^n)$ for $k = 4$, $O(1.93^n)$ for $k = 5$, etc. Using an algorithm based on the satisfiability coding lemma presented by Paturi, Publak, and Zane [10], the complexity starts higher at $O(1.78^n)$ for $k = 3$, but approaches 2^n more slowly as k increases (first exceeding $O(1.9^n)$ when $k = 8$). More recently, Dantsin et al. [4] published an algorithm that combines local search with branching on clauses to achieve lower complexities, specifically $O((2-2/(k+1))^n)$, reaching $O(1.9^n)$ when $k = 19$. Empirical studies using randomly generated expressions suggest that average-case time complexity is lower than worst-case, but still exponential and approaching 2^n as k increases. The analysis presented by Beame, Karp, Pitassi, and Saks [2] using a DLL resolution algorithm [5] places the expected time complexity for $k = 3$ at $O(1.15^n)$, reaching $O(1.9^n)$ at $k = 23$. Building on the satisfiability coding lemma, Impagliazzo and Paturi [7] present an analytical argument that the complexity of k -SAT increases with k . Under the assumption that SAT does not have a subexponential-time algorithm, they use expression decompositions and reductions to show that k -SAT is $O(2^{s_k n})$, where the constants s_k form an increasing sequence with $0 < s_k < 1$.

This paper discusses the complexity of k -SAT without reference to any specific satisfiability algorithm. In the conventional language-theoretic approach to satisfiability, the language studied is a set of strings representing Boolean expressions such as k -CNF or a set of constraint satisfaction formulas (a,b) -CSP, where each variable has a possible values and each constraint mentions at most b variables [6]. Here we consider Boolean expressions to be representations for sets of bit strings that correspond to satisfying assignments. The underlying language is a set of bit strings, and the expression is an acceptor that determines membership in the set. With k -CNF or $(2,k)$ -CSP, all expressions with differing numbers of variables are in the same language. Here we consider k -CNF(n) to be a family of language classes where the number of variables n is parameterized along with k . The strings in the languages are not Boolean expressions, but rather bit-string representations of the assignments that satisfy Boolean expressions. This approach has been used in the study of the complexity of circuits for finite Boolean functions [3]. The study of circuit complexity leads to a quest for a proof that some NP-complete function, such as a characteristic function for CLIQUE, does not have a polynomial circuit. Here we deal not with circuits and the functions they implement, but with normal form logical expressions and the languages they represent. When we systematically examine what languages are representable by k -CNF expressions over n variables, a proper hierarchy of languages emerges. Expressions with larger k are capable of representing larger classes of languages.

2 The Language Hierarchy

There is a natural correspondence between bit strings and assignments to a set of Boolean variables. We use B to represent the bit set $\{0, 1\}$. Let *false* correspond to 0 and let *true* correspond to 1. Place the variables in an ordered set $\{v_1, v_2, \dots, v_n\}$ and let the index i of each variable in the set correspond to bit position i in the corresponding string. An assignment to a set of three variables $v_1 = \text{true}$, $v_2 = \text{false}$, $v_3 = \text{false}$, for example, corresponds to the bit string 100. We will use $\text{bitstring}(A)$ to denote the bit string associated with assignment A , and $\text{assignment}(w)$ will denote the assignment corresponding to bit string w .

There is also a natural correspondence between sets of bit strings and instances of the satisfiability problem for Boolean expressions. We simply define the language of a Boolean expression to be the set of bit strings that represent the assignments that satisfy the expression. An expression over n variables is satisfiable if its language is non-empty.

Definition 2.1: The language of a Boolean expression E over n variables, denoted $L(E)$, is the subset of B^n containing all the strings w such that $\text{assignment}(w)$ satisfies E .

For two expressions E and F , we observe that $L(E) \cap L(F) \equiv L(E \wedge F)$. Using n and k as parameters, we can define classes of languages corresponding to categories of normal-form Boolean expressions. We understand the conventional notation k -CNF to represent the set of conjunctive normal form Boolean expressions with at most k literals per clause. We will assume that within a clause, each literal mentions a distinct variable. The set of CNF expressions is divided into subsets denoted k -CNF(n).

Definition 2.2: Let k -CNF(n) represent the set of conjunctive normal form Boolean expressions over n variables with zero or more clauses and at most k literals per clause, for n and $k > 0$.

Given the assumptions excluding redundancy in clauses, there is a finite number of distinct clauses for each n and k . As a result, each set $k\text{-CNF}(n)$ is a finite set of expressions, and $k\text{-CNF}$ is an infinite set of expressions containing the union of the $k\text{-CNF}(n)$ sets for all values of n . In some research literature, a $k\text{-CNF}$ expression is defined to have exactly k literals per clause. We point out that if a language is $L(E)$ for an expression in $k\text{-CNF}(n)$, it is also $L(E')$ for some expression in exact $k\text{-CNF}(n)$. The expression E' is built by replacing the short clauses in E with conjunctions of clauses with k literals. A simple example makes the case. If an expression $3\text{-CNF}(n)$ contains a clause of one variable v , choose any two other variables u and x from the variable set and replace the clause v with a conjunction of four clauses containing v and every possible pattern of negation over u and x : $(v \vee u \vee x) \wedge (v \vee u \vee \neg x) \wedge (v \vee \neg u \vee x) \wedge (v \vee \neg u \vee \neg x)$. This replacement does not affect the language of the expression. We now define a notation for the classes of languages corresponding to $k\text{-CNF}(n)$ expressions.

Definition 2.3: Let $X_n^k = \{L(E) : E \in k\text{-CNF}(n)\}$ for $n, k > 0, n \geq k$.

The proposition below establishes a fundamental property for the X_n^k classes that will be useful in proving other propositions, including the hierarchy theorem.

Proposition 2.1: Let $L = L(E)$ where E is a Boolean expression in $k\text{-CNF}(n)$. $\bar{L} = B^n - L$ is the union of m subsets of B^n where each subset $S_i, 1 \leq i \leq m$, contains 2^{n-k} strings that are all identical in k bit positions, and where m is the number of clauses in an exact $k\text{-CNF}(n)$ expression E' such that $L(E') = L(E)$.

Proof: For any clause p in an expression E , there is a subset of B^n corresponding to assignments that do not satisfy p , and \bar{L} is just the union of those subsets. First we create E' , an exact $k\text{-CNF}(n)$ expression such that $L(E') = L(E)$. Then each clause has k literals, and it is not satisfied by any of the 2^{n-k} assignments that contradict all of the literals in the clause. The strings corresponding to these assignments are therefore the same in k bit positions, and they have every possible pattern of 1's and 0's in the other $n-k$ bit positions. \square

The proposition above leads to a number of other results, culminating with the hierarchy theorem. First we show that some languages are found in all the X_n^k classes.

Proposition 2.2: X_n^k contains a) the empty language ϕ , b) the set of all strings B^n , and c) any subset S of B^n where $|S| = 1$, for $n, k > 0$ and $n \geq k$.

Proof: a) For any n and k , we can build an unsatisfiable expression by including all possible clauses, or more concisely, all possible clauses over any selection of k variables. No assignment can satisfy such an expression, so each X_n^k class contains ϕ . b) Also, for any n and k , an expression can contain zero clauses. By convention, any assignment satisfies all the clauses of such an expression, so each X_n^k class contains B^n . c) For any X_n^k class it is possible to build an expression E that has only one satisfying assignment A . Set E equal to the set of all possible clauses for n and k , and remove every clause that contradicts A in every one of its literals. The assignment A will satisfy every remaining clause, and every other assignment will fail to satisfy at least one of the remaining clauses. \square

Proposition 2.3: $X_n^n = \mathcal{P}(B^n)$.

Proof: Each clause in an expression E has n literals, and there is only one assignment that does not satisfy it. Given a language L , construct its expression E by adding a clause c to exclude each string w that is not in L . To accomplish this, literal v_i is added to clause c if the i^{th} bit of w is 0, and $\neg v_i$ is added if the i^{th} bit of w is 1. \square

And finally, a strict hierarchy emerges.

Proposition 2.4: X_n^k is a proper subclass of X_n^{k+1} for $n > 1$ and $1 \leq k < n$.

Proof: Consider the size of the languages of expressions that have exactly one clause. If an expression E in k -CNF(n) has one clause, its language has at most $2^n - 2^{n-k}$ strings. And if an expression has more clauses, its language is smaller yet. So all languages with $2^n - 2^{n-(k+1)}$ strings (and there are many such languages in X_n^{k+1}) are bigger than any proper subset of B^n in X_n^k .

It is easy to establish some basic closure properties for the X_n^k classes.

Proposition 2.5: X_n^n is closed under complement, union, and intersection for $n > 0$.

Proof: X_n^n is equivalent to $\mathcal{P}(B^n)$, which is closed under these operations. \square

Proposition 2.6: X_n^k is closed under intersection, for $n > 0$ and $1 \leq k \leq n$.

Proof: This follows immediately from the observation that for two expressions E and F , $L(E) \cap L(F) \equiv L(E \wedge F)$.

Proposition 2.7: X_n^k is not closed under complement and union, for $n > 0$ and $1 \leq k < n$.

Proof: We can use these operations to build languages that are known to be outside the class. With regard to union, we could build any language by repeated union of singleton languages. Regarding complement, we observe that the complement of a singleton language is bigger than any proper subset of B^n in X_n^k . \square

Since the ordering of the variables in an expression is arbitrary, it can easily be shown that the languages in the X_n^k classes are closed under uniform permutations. Let $\sigma: \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ be a bijection that implements some permutation of the natural numbers from 1 to n . Then we define the permutation of a string as $permute(w, \sigma) = w'$ such that $w'_{\sigma(i)} = w_i$, where w_i is the i^{th} symbol in w . The permutation of a language is defined as $permute(L, \sigma) = \{w: permute(w, \sigma) \in L\}$.

Proposition 2.8: X_n^k is closed under language permutations.

Proof: A permutation is implemented by reordering the variables. \square

3 Reductions among Languages in the Hierarchy

Two standard methods from the literature on Boolean satisfiability/tautology can be recast as reductions between languages within the hierarchy. Cook first published a reduction from DNF TAUTOLOGY to establish that 3-DNF TAUTOLOGY is an NP-complete problem [1]. The method of this reduction is applied here to define a reduction from a language in X_n^{k+1} to a language in X_n^k .

Definition 3.1: The reduction $R_f: (k+1)\text{-CNF}(n) \rightarrow k\text{-CNF}(n')$ is defined for $k \geq 3$. k -CNF expression E' with variable set V' is built from $(k+1)$ -CNF expression E with variable set V as follows:

Set $V' = V$.

For each clause c in E where c contains k or fewer literals,

conjoin c to E' .
 For each clause in E with $k+1$ literals $(l_1 \vee l_2 \vee \dots \vee l_{k+1})$,
 Create a new variable v and insert it at the end of V' .
 Conjoin clauses $(l_1 \vee l_2 \vee \dots \vee l_{k-1} \vee v)$ and $(l_k \vee l_{k+1} \vee \neg v)$ to E' .

In the construction of E' above, if expression E has m clauses including r clauses with $k+1$ literals, then E' has $n+r$ variables and $m+r$ clauses, and no clause contains more than k literals. The reduction is clearly accomplished in linear time, with $|E'| < 2 \cdot |E|$. In order to relate $L(E)$ with $L(E')$, we define a prefix operation on languages.

Definition 3.2: The set of prefixes of a language is defined as $prefix(L, n) = \{x \mid x \text{ is a string of length } n \text{ and for some } y, xy \in L\}$.

Proposition 3.1: For an expression E in $(k+1)$ -CNF(n), $L(E) = prefix(L(R_1(E)), n)$.

Proof: Every assignment that satisfied the original expression can be extended with some assignment to the new variables to satisfy the modified expression. First we show that $x \in L(E) \Rightarrow xy \in L(R_1(E))$ for some y . Since $assignment(x)$ satisfies every clause in E , it also satisfies those clauses in $R_1(E)$ that came directly from E . Consider a pair of clauses $c_1 = (l_1 \vee l_2 \vee \dots \vee l_{k-1} \vee v)$ and $c_2 = (l_k \vee l_{k+1} \vee \neg v)$ that were added to $R_1(E)$ as a replacement for a clause $c = (l_1 \vee l_2 \vee \dots \vee l_{k+1})$ in E . The variable v is a new variable with index $> n$ that occurs only in c_1 and c_2 . Since $assignment(x)$ satisfies c , we know that it will also satisfy either c_1 or c_2 . If it satisfies c_1 , then we set the variable v to the truth value that will satisfy c_2 , and if $assignment(x)$ satisfies c_2 , we set v to satisfy c_1 . Each variable v_i with $i > n$ represents such a pair of clauses, and its value is assigned to satisfy the clause in the pair that is not satisfied by $assignment(x)$. These assignments define the bit string y such that $assignment(xy)$ satisfies $R_1(E)$. The proof in the opposite direction is similar. \square

The reduction R_1 can be applied iteratively to convert an expression in k -CNF(n) to another expression in k' -CNF(n') for any $k > k' \geq 3$. The reduction is linear time for each iteration, and it remains linear time overall in contexts where $\Delta k = k - k'$ is strictly constant. The reduction cannot be used to cross the boundary from 3-CNF to 2-CNF, since it always introduces new clauses of width 3.

Another reduction R_2 can be defined to convert an expression in k -CNF(n) to an expression in $(2k-2)$ -CNF($n-1$) with one less variable but more and wider clauses. The method of R_2 is inversely related to the method of R_1 . In the literature of automated theorem proving, it is also known as logical resolution [11].

Definition 3.3: The reduction $R_2: k\text{-CNF}(n) \rightarrow (2k-2)\text{-CNF}(n-1)$ is defined for $k > 1$. Expression E' with variable set V' is built from k -CNF expression E with variable set V as follows:

Set $V' = V - \{v_n\}$.
 For each clause c in E where c does not mention v_n ,
 conjoin c to E' .
 For each pair of clauses $(l_1 \vee l_2 \vee \dots \vee l_i \vee v_n)$ and $(m_1 \vee m_2 \vee \dots \vee m_j \vee \neg v_n)$ in E ,
 Conjoin clause $(l_1 \vee l_2 \vee \dots \vee l_i \vee m_1 \vee m_2 \vee \dots \vee m_j)$ to E' .

The R_2 reduction decreases the number of variables by one while increasing the maximum number of literals per clause from k to $2(k-1)$. The proposition below relates $L(E)$ to $L(R_2(E))$.

Proposition 3.2: For an expression E in k -CNF(n), $L(R_2(E)) = \text{prefix}(L(E), n-1)$.

Proof: First we show that $xb \in L(E) \Rightarrow x \in L(R_2(E))$ where $b = 0$ or $b = 1$. Since $\text{assignment}(xb)$ satisfies every clause in E , it also satisfies those clauses in $R_2(E)$ that came directly from E . Also, for any clause $c = (l_1 \vee l_2 \vee \dots \vee l_i \vee m_1 \vee m_2 \vee \dots \vee m_j)$ in $R_2(E)$ that is not in E , we observe that E must contain clauses $(l_1 \vee l_2 \vee \dots \vee l_i \vee v_n)$ and $(m_1 \vee m_2 \vee \dots \vee m_j \vee \neg v_n)$, both satisfied by $\text{assignment}(xb)$. The bit b represents the value of v_n , and it is clear that b cannot be used to satisfy both clauses. We conclude that $\text{assignment}(x)$ must satisfy at least one of $\{l_1, l_2, \dots, l_i, m_1, m_2, \dots, m_j\}$, thus satisfying clause c .

We must also show that $x \in L(R_2(E)) \Rightarrow xb \in L(E)$ where $b = 0$ or $b = 1$. First we observe that $\text{assignment}(x)$ satisfies every clause in E that also occurs in $R_2(E)$. This leaves only the clauses in E that contain v_n or $\neg v_n$. Let A be the set of clauses $\{\alpha: \alpha \vee v_n \in E\}$, and let $B = \{\beta: \beta \vee \neg v_n \in E\}$. It must be true that $\text{assignment}(x)$ satisfies all clauses in A or all clauses in B . If this is not true, then there are clauses $\alpha \in A$ and $\beta \in B$ such that $(\alpha \vee \beta)$ is in $R_2(E)$ but is not satisfied by $\text{assignment}(x)$. If $\text{assignment}(x)$ satisfies all clauses in A , we set $b = 0$. Otherwise, we set $b = 1$. Then $\text{assignment}(xb)$ satisfies all clauses in E . \square

4 Relevance of the Hierarchy

The X_n^k hierarchy provides a context for the classification, comparison, and development of algorithms for CNF satisfiability. More generally, it provides a framework for comparing the complexities of languages that have been classified as NP-complete. We believe that representational and algorithmic complexity are interconnected, and that understanding the underlying representational issues is a key step in determining any problem's complexity. Stearns and Hunt [12] have defined power indices as a measure of complexity among NP-complete problems, arguing that problems with lower power index have lower (although still exponential) complexity. The X_n^k hierarchy provides another kind of evidence that some NP-complete problems are simpler than others. With reference to satisfiability, the X_n^k hierarchy with $k \geq 3$ presents a refinement of power index one. Boolean expressions and the languages they represent become increasingly complex as k increases, and we expect algorithms for the same problem (such as satisfiability) across these language classes to require more time at higher levels.

Stearns and Hunt pointed out that it makes little sense to apply a linear reduction to transform a problem with lower power index to one with higher power index. Thus reductions from CLIQUE to other problems with power index one are not advisable. Applying the same logic in the context of representational complexity, we should avoid reductions from X_n^k to $X_n^{k'}$, where $k' > k$. This implies that logical resolution (reduction R_2 above) is not likely to be the best algorithm for satisfiability testing. Even the inverse reduction (R_1 above), which moves a problem in the opposite direction, increases the number of variables too rapidly to guarantee an improvement in time complexity. Thus transformations among the X_n^k classes that preserve conjunctive normal form are not likely to produce improved upper bounds on time complexity. Resolution can also be characterized as a backtracking search that branches on variables. Historically, as discussed in section 1 above, worst-case complexity improvements are derived from algorithms that branch on clauses, not variables. While these algorithms are commonly perceived to be resolution-based, they actually transform CNF expressions to an intermediate logical form which is neither conjunctive nor disjunctive normal form (see [9]), avoiding the resolution-based "reduction" to a more complex language class.

5 References

- [1] S. Cook, "The Complexity of Theorem-Proving Procedures," Proceedings of the Third ACM Symposium on Theory of Computing, ACM, New York, pp. 151-158, 1971.
- [2] P. Beame, R. Karp, T. Pitassi, and M. Saks, "On the Complexity of Unsatisfiability Proofs for Random k -CNF Formulas," Proceedings of the 30th ACM Symposium of the Theory of Computing (STOC 98), Dallas, Texas, pp. 561-571, 1998.
- [3] R. Boppana and M. Sipser, "The Complexity of Finite Functions," in *The Handbook of Theoretical Computer Science*, vol. 1: Algorithms and Complexity, ed. By J. van Leeuwen, MIT Press, Cambridge, Mass., pp. 757-804, 1990.
- [4] E. Dantsin, A. Goerdt, E. Hirsh, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schoning, "A Deterministic $(2-2/(k+1))^n$ Algorithm for k -SAT based on Local Search," *Theoretical Computer Science* 289:1, pp. 69-83, 2002.
- [5] M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem-proving," *Communications ACM* 5, pp. 394-397, 1962.
- [6] D. Eppstein, "Improved Algorithms for 3-Coloring, 3-Edge-Coloring, and Constraint Satisfaction," Proceedings of the 12th Symposium on Discrete Algorithms, pp. 329-337, 2001.
- [7] R. Impagliazzo and R. Paturi, "Complexity of k -SAT," Proceedings of the IEEE Conference on Computational Complexity, pp. 237-240, 1999.
- [8] B. Monien and E. Speckenmeyer, "Solving Satisfiability in Less Than 2^n Steps," *Discrete Applied Mathematics* 10, pp. 287-295, 1985.
- [9] T. E. O'Neil, "A Clause-Based Reduction from k -SAT to CLIQUE," Proceedings of the 2005 International Conference on Foundations of Computer Science (FCS '05), Las Vegas, Nevada, pp. 17-23, June 2005.
- [10] R. Paturi, P. Pudlak, and F. Zane, "Satisfiability Coding Lemma," Proceedings 38th Symposium on Foundations of Computer Science, IEEE, pp. 566-574, 1997.
- [11] J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *Journal of the Association for Computing Machinery* 12(1), pp. 23-41, 1965.
- [12] R. Stearns and H. Hunt, "Power Indices and Easier Hard Problems," *Mathematical Systems Theory* 23, pp.209-225, 1990.