Prime implicants of CNF formulas

September 8, 2020

Abstract

We prove that every n-variable k-CNF formula has at most $n \cdot 3^{n(1-\Omega(1/k))}$ prime implicants. This is near optimal, almost matching the best known lower bound of $3^{n(1-O(\log k)/k)}$. This resolves an open problem from the Ph.D. thesis of Talebanfard, who introduced the question and gave the first non-trivial bounds for small-read k-CNF formulas.

Our proof technique is algorithmic in nature, yielding an algorithm for computing the set of all prime implicants—the *Blake Normal Form*—of a given k-CNF formula. The problem of computing the Blake Normal Form of a given function is a classic one, dating back to Quine, and our work gives the first non-trivial algorithm for k-CNF formulas.

At the heart of our analysis is a generalization of the *Satisfiability Coding Lemma* of Paturi, Pudlák, and Zane. This lemma gives a coding scheme for satisfying assignments of k-CNF formulas; we generalize it to give a coding scheme for implicants.

1 Introduction

Given a boolean function φ, an *implicant* of φ is a set of literals I whose conjunction implies φ. That is every assignment that satisfies all the literals in I satisfies φ. A *prime implicant* is an implicant, I, such that no strict subset of I is an implicant. In other words, when viewing the satisfying assignments of an n-variable formula as a subgraph of the n-dimensional hypercube graph, a prime implicant of size j is a j-dimensional cube in the graph that cannot be extended to a $(j+1)$-dimensional cube.

In this work, we study two of the most basic questions related to prime implicants, the first structural and the second algorithmic:

- ○ First, how many prime implicants can a function have?
- ○ Second, can one quickly enumerate all prime implicants of a given function?

Historically, both questions have received a great deal of interest. With respect to the first question, extremal bounds for the class of all n-variable functions are fairly well-understood. Let $\#\mathrm{PrimeImplicants}(\varphi)$ denote the number of prime implicants of a function φ. A classic result of Chandra and Markowsky shows a universal bound of $\#\mathrm{PrimeImplicants}(\varphi) \leq O(3^n/\sqrt{n})$ for every n-variable boolean function φ [CM78]. Even earlier work of Dunham and Fridshal provides a near-matching lower bound, constructing a n-variable function that has $\Omega(3^n/n)$ prime implicants [DF59].

Turning to the second question, the algorithmic problem of obtaining the disjunction of all prime implicants of a given function φ, called the *Blake Canonical Form* (BCF) of φ, is also well-studied. Unlike a minimal equivalent DNF, the BCF of a formula is unique. Schaefer and Umans showed obtaining a minimal DNF to be Σ_2-complete [SU02], and a classic and well-known approach to obtaining a minimal DNF, the *Quine-McClusky algorithm* [Qui52, Qui55, McC56], proceeds by generating and then paring down the BCF. This algorithm obtains the BCF by drawing up a table of satisfying assignments and then finding prime implicants by iterated consensus. Alternative methods of obtaining the BCF include reduction to integer or 0-1 programming [MFSO97, PPP99] and the use of SAT-solvers, directly or through a generalization [JMSSS14, DFLBM13]. However, none of these approaches offers any runtime guarantees better than the trivial bound of $O(|\varphi| \cdot 3^n)$ associated with an exhaustive search, where $|\varphi|$ denotes the representation size of φ.

This work: Prime Implicants of k-CNF formulas. The main contributions of our work are near-optimal answers to both questions in the case where φ is represented as a k-CNF formula. The problem of bounding the number of prime implicants for k-CNF formulas was first considered in the Ph.D. thesis of Talebanfard [Tal14], who gave a non-trivial upper bound for *read-r* k-CNF formulas, i.e. those in which each variable occurs in at most r clauses:

Theorem 1 ([Tal14]). *Let φ be an n-variable read-r k-CNF formula. Then* $\#\mathrm{PrimeImplicants}(\varphi) \leq 3^{n(1-1/(rk))}$.

For constant-read k-CNFs, this gives a bound of $3^{n(1-\Omega(1/k))}$. In the conclusion of his thesis, Talebanfard asked whether this bound in fact holds for all k-CNF formulas, with no restriction on read. Our first main result answers Talebanfard's question in the affirmative:

Theorem 2. *There is a universal constant $c > 0$ such that the following holds. For all n-variable k-CNF formulas φ,*

$$\#\text{PrimeImplicants}(\varphi) \leq n \cdot 3^{n(1-c/k)}.$$

Theorem 2 is near optimal, as [Tal14] also shows the existence of a k-CNF formula with $3^{n(1-O(\log k)/k)}$ prime implicants (the construction of this formula is based on the lower bound construction of [DF59] discussed above).

Our proof of Theorem 2 is algorithmic in nature, and the same proof technique also yields our second main result, the first non-trivial algorithm for enumerating the prime implicants of a given k-CNF formula:

Theorem 3. *There is a universal constant $c > 0$ such that the following holds. There is a deterministic algorithm which, given as input an n-variable k-CNF formula φ, computes the Blake Canonical Form of φ in time*

$$O(|\varphi| \cdot n^{2k+2} \cdot 3^{n(1-c/k)}).$$

The runtime of this algorithm is similarly near optimal, given the aforementioned existence of a k-CNF formula with $3^{n(1-O(\log k)/k)}$ many prime implicants.

Our technique: A generalization of the [PPZ97] Satisfiability Coding Lemma. In [Tal16], Talebanfard remarked that in order to answer his question, one would need to develop a generalization of the [PPZ97] Satisfiability Coding Lemma that "can treat isolated solutions and prime implicants in general as the same objects". As we now illustrate, this is exactly what we do to obtain our results.

To introduce a key definition, fix a k-CNF φ and view an implicant I as a partial solution; for $i \in [n]$,

$$I(x_i) = \begin{cases} 0 & \text{if } \neg x_i \in I \\ 1 & \text{if } x_i \in I \\ * & \text{otherwise.} \end{cases}$$

Variables assigned 1 or 0 are *fixed* by I and variables assigned $*$ are *free*. A clause C of φ is *critical* for variable x_i if I maps each literal in C to 0 or $*$, except for the literal involving x_i. Thus if I is total and C is critical for x_i, flipping the value of x_i makes I into a falsifying assignment; if I is instead partial, this holds for some total solution J agreeing with I on I's fixed variables.

A solution is *j-isolated* if j variables have critical clauses. The seminal Satisfiability Coding Lemma due to Paturi, Pudlák and Zane [PPZ97] gives an encoding for *total* j-isolated solutions:

Lemma 1.1 (Satisfiability Coding Lemma [PPZ97])**.** *Fix a total, j-isolated solution I of a k-CNF φ on n variables. There exists a randomized, one-to-one, prefix free encoding $\text{ENC}^k_{\text{PPZ}}(I)$ over the alphabet $\{0,1\}$ with expected length at most $n - j/k$.*

At a high level, the encoding $\text{ENC}^k_{\text{PPZ}}$ iterates through the randomly permuted variables, writing down their values in the encoding, and plugging in $I(x_i)$ for x_i in φ on iteration i. This simplifies φ with each iteration, so that a unit clause may appear in φ, in which case the value for x_i is not written down in the encoding; this saves precious space. Our encoding does exactly this, simply deleting literals containing x_i when $I(x_i) = *$ and appending the value $*$ of x_i to the encoding. Thus when run on total solutions, the encodings ENC^k and $\text{ENC}^k_{\text{PPZ}}$ agree exactly; we generalize the Satisfiability Coding Lemma to *partial* j-isolated implicants:

Lemma 1.2 (Implicant Coding Lemma)**.** *Fix a j-isolated implicant I of a k-CNF φ on n variables. There exists a randomized, one-to-one, prefix free encoding* $\mathrm{ENC}^k(I)$ *over the alphabet* $\{0, 1, *\}$ *with expected length at most $n - j/k$.*

The Satisfiability Coding Lemma [PPZ97] yields a bound on the number of j-isolated total solutions and a $O(|\varphi| \cdot n^{3k} \cdot 2^{n(1-\Omega(1/k))})$-time deterministic algorithm for k-SAT; beating $\mathrm{poly}(k, n) \cdot 2^{n(1-\Omega(1/k))}$ remains a central open problem in complexity theory. Similarly, Theorems 2 and 3 follow as straightforward consequences of the above lemma and the observation that a size-j prime implicant is j-isolated. We conclude with extensions of these results to m-clause and monotone CNFs.

Extensions: m-clause and monotone CNFs. We extend our results to m-clause and monotone CNF formulas. We first obtain a natural extension for m-clause formulas:

Theorem 4. *There is some universal constant c such that for any m-clause CNF φ,*

$$\#\mathrm{PrimeImplicants}(\varphi) \leq n \cdot 3^{n(1-c/\log m)}.$$

Theorem 5. *There are some universal constants c, c' such that for any m-clause formula φ we can deterministically obtain the BCF in time*

$$O(|\varphi| \cdot n^{2 \cdot c' \cdot \log m + 2} \cdot 3^{n(1-c/\log m)}).$$

For monotone formulas, our construction gives a sharper bound and a faster algorithm:

Theorem 6. *There is some universal constant c such that for any monotone k-CNF φ,*

$$\#\mathrm{PrimeImplicants}(\varphi) \leq n \cdot 2^{n(1-c/k)}$$

Theorem 7. *There is some universal constant c such that for any monotone k-CNF φ, we deterministically obtain the BCF in time*

$$O(|\varphi| \cdot n^{2k+2} \cdot 2^{n(1-c/k)}).$$

2 Implicant Encoding Lemma

In this section, we prove Lemma 1.2. We'll first state our encoding and decoding algorithms in plain English, following up with formal presentations and a proof of the lemma.

The encoder $\mathrm{ENC}^k_\pi(\varphi, I)$ is parameterized by a random permutation π, which it uses to relabel the variables in φ and I. We initialize the encoding to be the empty string and then, for each $i \in [n]$, do the following. If x_i is in some unit clause of φ, do nothing. Otherwise, we append the value assigned to x_i to the encoding. We then replace φ with $\varphi(i, I)$, the result of substituting $I[i]$ for x_i in φ. Thus if x_i is free, all literals containing it are deleted, and if x_i is fixed, all literals which contain x_i and are assigned 0 are deleted, and all other literals containing x_i are assigned 1, so that their clauses are satisfied and so deleted. Formally:

Algorithm 1: Encoding algorithm ENC^k_π for k-CNFs

```
relabel $\varphi$ and $I$ according to $\pi$
$\varphi_0 = \varphi$
$y =$ “”
for $i$ in 1,...,$n$ do
    if $x_i$ not in a unit clause in $\varphi_{i-1}$ then
        $y \mathrel{+}= I[i]$
    end
    $\varphi_i = \varphi_{i-1}(i, I[i])$
end
return $y$
```

The decoder $\mathrm{DEC}^k_\pi(\varphi, y)$ again relabels φ according to π, and then for each $i \in [n]$ does the following. Simply read the value for x_i off of the encoding y, as long as there is no unit clause. If there is a unit clause containing a literal x_i or $\neg x_i$, the decoder sets the variable x_i to satisfy the clause, instead of reading another value from the encoding. In both cases, the decoder assigns the variable x_i and simplifies the formula φ before proceeding to the next iteration. Formally:

Algorithm 2: Decoding algorithm DEC^k_π for k-CNFs

```
relabel $\varphi$ according to $\pi$
$\psi_0 = \varphi$
$J$ = empty assignment
for $i$ in 1,...,$n$ do
    if $x_i$ not a unit clause of $\psi_{i-1}$ then
        $J[i] =$ read next bit of $y$
    else
        $J[i] = 1$ if unit clause is $x_i$ and 0 if the unit clause is $\neg x_i$
    end
    $\psi_i = \psi_{i-1}(i, J[i])$
end
Relabel $J$ according to $\pi^{-1}$
return $J$
```

We now prove Lemma 1.2:

Proof. **One-to-one and prefix-free.** To show this, we'll show that the decoding algorithm inverts the encoding algorithm. That is, $\mathrm{DEC}^k_\pi(\varphi, \mathrm{ENC}^k_\pi(\varphi, I)) = I$ for all j-isolated implicants I of φ. Fix such an implicant I and its encoding $y = \mathrm{ENC}^k_\pi(\varphi, I)$. It suffices to show by induction that the following hold at the beginning of iteration i.

1. the restriction of I to the first $i-1$ variables is precisely the value of J constructed by $\mathrm{DEC}^k_\pi(\varphi, y)$ so far.

2. the encoder and decoder have inspected the same number of characters in the encoding, moving from left to right.

3. $\varphi_{i-1} = \psi_{i-1}$.

The base case holds trivially. Assuming the claims hold at the start of iteration i, we'll prove the claims hold at the start of for iteration $i+1$. Consider the ith iteration. By the inductive hypothesis $\varphi_{i-1} = \psi_{i-1}$, so both the encoding and decoding algorithms move into the same case, depending on the presence of a unit clause.

In the if case, the encoder adds the encoding precisely the value used by the decoder, by the inductive hypothesis on the second claim. This establishes the first claim. Both the encoder and decoder use one character of the encoding, so the second claim holds as well. The third claim then follows from the first claim, and the fact that $\varphi_{i-1} = \psi_{i-1}$.

In the else case, since I is an implicant, only one of x_i and $\neg x_i$ can be a unit clause in $\psi_{i-1} = \varphi_{i-1}$. Therefore this uniquely (and correctly) determines the value used by the decoder. This establishes the first claim. Neither the encoder nor the decoder use a character of the encoding, so the second claim holds as well. Finally, the third claims follows from the first claim and the inductive hypothesis on the third claim

Therefore at the end of iteration i, or the beginning of iteration $i+1$, each of the claims hold, completing the induction.

Encoding length. It suffices to show that for any j-sensitive implicant I, the expected encoding length of I under ENC_π^k over uniformly random permutations π is at most $n-j/k$. For then, in particular, there is some π for which ENC_π^k is a prefix free encoding with expected encoding length at most $n-j/k$.

Let I be any j-sensitive implicant of φ, and π be a uniformly random permutation on $[n]$. If in the ordering after application of π, a variable x_i comes after all other variables in its critical clause, ENC_π^k skips that variable. The probability that this occurs is at least $1/k$, there being at most k variables in the clause and π being uniform. Since I is j-isolated, there are at least j critical clauses, so that by linearity of expectation, at least j/k fixed variables are skipped, giving an encoding of expected length at most $n-j/k$. □

It will serve later discussion to have a bound on the runtime of these algorithms:

Lemma 2.1. *ENC_π^k and DEC_π^k each run in time $O(|\varphi| \cdot n)$.*

Proof. We consider only ENC_π^k, as the proof for DEC_π^k is exactly the same. It suffices to show that each of the iterations i for $i \in [n]$ takes time $O(|\varphi|)$. On iteration i, we attempt to determine whether the variable x_i appears in a unit clause of φ_{i-1}, and update the formula. Each of these can be done with a linear scan through the formula, requiring time $|\varphi_{i-1}| \leq |\varphi|$. □

3 Bounding the Number of Prime Implicants for k-CNFs

Let φ be any k-CNF. In this section, we use our encoding lemma to give a new upper bound on #PrimeImplicants(φ). Our bound implies that for all n and all $k \geq 10$,

$$\#\mathrm{PrimeImplicants}(\varphi) \leq n \cdot 3^{n(1-1/(2k))}.$$

Our bound requires the following key fact:

Fact 3.1. *If ENC is a one-to-one, prefix-free encoding of S into strings formed from the alphabet {0,1,*}, and ENC has average code length ℓ, then $|S| \leq 3^\ell$.*

Proof. Let ℓ_I denote the length of $\text{ENC}(I)$ for $I \in S$. Then $\ell = \sum_{I\in S} \ell_I/|S|$, is the average encoding length. Since ENC is one-to-one and prefix-free, events E_I of rolling a three-sided die and stopping upon generating I are disjoint, and their probabilities sum to 1: $\sum_{I\in S} 3^{-\ell_I} \le 1$. We wish to show that $\ell \ge \log_3 |S|$:

$$\begin{aligned}\ell - \log_3 |S| &= \sum_{I\in S} \frac{1}{|S|}(\ell_I - \log_3 |S|) = -\sum_{I\in S} \frac{1}{|S|}(\log_3 3^{-\ell_I} + \log_3 |S|) \\ &= -\sum_{I\in S} \log_3(|S|3^{-\ell_I}) \ge -\log_3(\sum_{I\in S} 3^{-\ell_I}) \ge 0,\end{aligned}$$

where the penultimate inequality follows from concavity of log. □

Since size-j prime implicants are j-isolated, together with Lemma 1.2, the above fact immediately gives a bound on $\#\text{PrimeImplicants}_j(\varphi)$, the number of size-$j$ prime implicants of φ:

Lemma 3.2. *Let φ be a k-CNF. $\#\text{PrimeImplicants}_j(\varphi) \le 3^{n-j/k}$.*

We can now bound the total number of prime implicants for a k-CNF φ, proving Theorem 2:

Theorem 2. *Let φ be a k-CNF. Then for $c = \frac{1-\log_3(2)}{\log_3(2)+1/k}$,*

$$\#\text{PrimeImplicants}(\varphi) \le n \cdot 3^{n(1-c/k)}.$$

In particular, for all $k \ge 2$, one can take $c = .326$.

Proof of Theorem 2. Let φ be a k-CNF. We show that

$$\#\text{PrimeImplicants}_j(\varphi) \le 3^{n(1-c/k)} \text{ for } j \in [n].$$

Summing over j then gives the desired result. We consider two cases. If $j \ge nc$, by Lemma 3.2,

$$\#\text{PrimeImplicants}_j(\varphi) \le 3^{n-j/k} \le 3^{n-nc/k}.$$

Otherwise, $j < nc$. Since $\#\text{PrimeImplicants}_j(\varphi)$ is at most the number of ways of choosing locations for j fixed variables and then assigning them,

$$\#\text{PrimeImplicants}_j(\varphi) \le \binom{n}{j} \cdot 2^j < 2^{n+j} < 2^{n(1+c)}.$$

To conclude, note that $2^{n(1+c)} \le 3^{n(1-c/k)}$ precisely when

$$c \le (1 - \log_3(2))/(\log_3(2) + 1/k).$$ □

4 Obtaining Prime Implicants of k-CNFs

In this section, we give a deterministic algorithm for obtaining all the prime implicants of a k-CNF. We begin by noting that one can quickly check whether a given implicant is prime:

Lemma 4.1. *Fix a k-CNF φ. One can confirm in time $O(|\varphi| \cdot n)$ that an implicant I is a prime.*

Proof. Compute $\varphi(I)$, the result of performing all substitutions $\varphi(i, I)$ for $i \in [n]$; this takes time $O(|\varphi|)$. If some clause is empty, reject. Otherwise, if it is a critical clause, add the critical variable to a set S; this takes time $O(|\varphi| + n \cdot \log(n))$. Finally, check that S is contains precisely the variable fixed by I; this takes time $O(n \cdot \log(n))$. We'll only be needing the loose bound of $O(|\varphi| \cdot n)$. $\square$

We now give an algorithm for finding all size-j prime implicants. At a high level, the idea is to try to decode every string s in $\{0, 1, *\}^{n-j/k+1}$ using every permutation π. This works, because for each size-j prime implicant I, there is some permutation π such that ENC_π^k maps I to a prefix of some such s, and the code is prefix-free. The only problem is that there are many permutations π; to make the algorithm faster, we'll pick a smaller set of permutations π that gives the same guarantee. We now rehearse a standard argument to show that this set of permutations exists.

Lemma 4.2. *Fix k and a prime power N. There exist random variables $X_1, ..., X_N$ uniformly distributed over N which are k-wise independent, defined over a probability space of size N^k.*

Proof. Let $\mathbf{F}$ be a finite field of N elements. Sample uniform $c_0, ...c_{k-1} \in \mathbf{F}$. Let $X_\alpha = \sum_i c_i \alpha^i$. There are N^k choices of coefficients. k-wise independence follows by Lagrange interpolation. $\square$

Lemma 4.3. *There exists a set Π of permutations of size $O(n^{2k})$ such that for any size-j prime implicant I, the expected encoding length over $\pi \in \Pi$ is at most $n - j/k + 1$.*

Proof. Let N be the smallest power of 2 greater than n^2. Use Lemma 4.3 to obtain $\mathbf{X} = \{X_1, ..., X_n\}$. The probability that X_1 is last among the first k variables is at least

$$\frac{1}{k}\mathbf{P}[X_1, ..., X_k \text{ distinct}] \geq \frac{1}{k}\left(1 - \frac{k^2}{N}\right),$$

the expression on the right following from a union bound over all $\leq k^2$ pairs X_i, X_j on the event that $X_i = X_j$, which occurs with probability $1/N$. Similar reasoning applies for any other choice of k of the random variables in $\mathbf{X}$. Letting $\pi_{\mathbf{X}}(i)$ be the number of j such that $X_j \leq X_i$, we get a permutation by breaking ties arbitrarily, such that the expected coding length with respect to the random variable $\pi_{\mathbf{X}}$ is at most

$$n - \frac{j}{k}\left(1 - \frac{k^2}{N}\right) = n - j/k + \frac{jk}{N^2} \leq n - j/k + 1.$$

Let Π contain all possible permutations $\pi_{\mathbf{X}}$, of which there are $N^k = O(n^{2k})$, by Lemma 4.2. $\square$

We now consider the faster version of the naive algorithm described above and bound its runtime:

Algorithm 3: Find size-j prime implicants of a k-CNF

```
PI = $\emptyset$
$\Pi$ = the permutations given by Lemma 4.3
for $\pi \in \Pi$ and $s \in \{0, 1, *\}^{n-j/k+1}$ do
    $I = \mathrm{DEC}_\pi^k(s)$
    if $I$ is a prime implicant then
        add $I$ to the set PI
    end
end
Return PI
```

Lemma 4.4. *The above algorithm runs in time $O(|\varphi| \cdot n^{2k+1} \cdot 3^{n-j/k})$.*

Proof. By Lemmas 2.1 and 4.1, we repeat $O(|\varphi| \cdot n)$ operations for $O(n^{2k} \cdot 3^{n-j/k})$ iterations. $\square$

We can now show Theorem 3:

Theorem 3. *Let φ be a k-CNF. Then for $c = \frac{1-\log_3(2)}{\log_3(2)+1/k}$, we deterministically obtain the BCF in time*

$$O(|\varphi| \cdot n^{2k+2} \cdot 3^{n(1-c/k)}).$$

Proof of Theorem 3. It suffices to show that one can find all size-j prime implicants in time $O(|\varphi| \cdot n^{2k+1} \cdot 3^{n(1-c/k)})$, since summing over $j \in [n]$ then gives the desired result. As in the proof of Theorem 2, we consider two cases. If $j \geq nc$, we use Algorithm 3. Otherwise, $j < nc$. Then we enumerate over all partial assignments with at most j fixed coordinates, and, using Lemma 4.1, check in linear time if each is a prime implicant. This takes time

$$|\varphi| \cdot \binom{n}{j} \cdot 2^j \leq |\varphi| \cdot 3^{n(1-c/k)},$$

by the same reasoning used in the proof of Theorem 2. $\square$

5 Extensions

In this section, we lay out extensions of our results to m-clause and monotone CNFs.

5.1 m-clause CNFs

We first extend the results of the previous section to m-clause CNFs φ. In this subsection, we set $k = \frac{2+\log_3(m)}{1-\log_3(2)}$. For a clause C denote by C^k the truncation of C to the first k variables, and denote by φ^k the conjunction of C^k for all clauses C in φ.

5.1.1 An Encoding Lemma

Drawing on Hirahara [Hir17], we give a coding lemma for m-clause CNFs:

Lemma 5.1. *Let φ be any m-clause CNF, and I be a size-j prime implicant of φ. Then Algorithm 4 is a randomized one-to-one, prefix free encoding such that the expected length of I is at most $n - j/k + 1$, where $k = \frac{2+\log_3(m)}{1-\log_3(2)}$.*

The encoding algorithm, ENC^m_π, is a recursive procedure that has two cases. In the first case, I is a prime implicant of φ^k. we return $1 \circ \mathrm{ENC}^k_\pi$. In the second case, I is not a prime implicant of φ^k. Here we select some clause C^k_i such that $I(C^k_i)$ does not have any literal assigned 1 (The existence of such a clause is argued in Lemma 5.1). We simplify the formula by assigning all the variables in C^k_i according to I and call the formula recursively on the simplified formula and the remaining partial assignment, prepending the result with $0 \circ \langle i \rangle_3 \circ \langle C^k_i \rangle_3$. Where $\langle \cdot \rangle_3$ is a canonical representation of $\cdot$ using ternary digits. Note that our choice in C_i^k ensures that $I(C_i^k) \in \{0, *\}^k$, which requires $k \log_3(2)$ ternary digits to represent.

Algorithm 4: Encoding algorithm ENC_π^m for m-clause CNFs

```
initialize an empty string $enc$
let $C_1, ..., C_{m'}$ enumerate the clauses in $\varphi$
if $I$ is a prime implicant of $\varphi^k$ then
    enc += $1 \circ ENC_\pi^k(\varphi^k, I)$
    break
else
    fix some clause $C_i^k$ such that $I(C_i^k)$ does not have a variable assigned 1
    let $S$ contain the variables appearing in $C_i^k$
    enc += $0 \circ i \circ \langle C_i^k \rangle_3 \circ ENC_\pi^m(\varphi(I|_S), I|_{\neg S})$
```

The decoding algorithm DEC_π^m inverts the encoding as follows. It reads the first bit to determine which case split was taken during the encoding, if it was 1, then simply return $\mathrm{DEC}_\pi^k(\varphi, y)$. If it was 0, read i and $I(C_i^k)$ from the next $\log_3(m)$ and $k \log_3(2)$ bits respectively. From this we can determine I's assignments to the k variables in C_i^k. Then,we simplify φ by with these assignments and recurse on the simplified formula and the remainder of the encoding.

Algorithm 5: Decoding algorithm DEC_π^m for m-clause CNFs

```
initialize an empty assignment $I$
case = read first bit of enc
if $case = 1$ then
    return $DEC_\pi^k(\varphi^k, y)$
else
    $i$ = read the next $\log_3(m)$ bits of $y$
    fill $I$ by the partial assignment defined by the next $k \log_3(2)$ bits
    recursively call $DEC_\pi^k(\varphi(I), y)$
```

Proof. **One-to-one and prefix free.** We claim that $\mathrm{DEC}_\pi^m(\varphi, \mathrm{ENC}_\pi^m(\varphi, I)) = I$ for any prime implicant I, so ENC_π^m is a prefix-free encoding. If there is no recursive call, this follows immediately from the result for ENC_π^k. The recursive case is clearly invertable; we now claim that in this case the desired clause exists. It suffices to show that I is not an implicant of φ^k. As I is not a prime implicant of φ^k, it is either not an implicant, or an implicant but not prime. We'll show that it can't be the latter. By contradiction, assume I is an implicant of φ^k but not prime. So there is some $J \subsetneq I$ such that J is still an implicant of I, but J is also an implicant of φ, which contradicts the primeness of I in φ. It is straightforward to check that $I|_{\neg S}$ is indeed a prime implicant of $\varphi(I|_S)$. Thus we recurse until we reach the base case.

Encoding length. We claim by induction on the number of recursive calls that the length of the encoding is in expectation (with respect to π) at most $n - j/k + 1$. If there is no recursive call, the encoding length is 1 more than the expected encoding length of ENC_π^k, which by Lemma 1.2 is at most $n - j/k$, as required.

If there is a recursive call, using the inductive hypothesis, the average length of the encoding when there is a recursive call is at most

$$\begin{aligned}&1+\log_3(m)+k\log_3(2)+[(n-k)-(j-k)/k+1]\\&=1+\log_3(m)+k\log_3(2)+n-k-j/k+2\\&=n-j/k+1,\end{aligned}$$

the last equality following from our choice of k, which gives $\log_3(m)+k\log_2(3)-k+2=0$. □

5.1.2 Bounding the Number of Prime Implicants

Let $\#\text{PrimeImplicants}_j(\varphi)$ denote the number of j-fixed prime implicants. As in the k-CNF case, we get a bound immediately from the encoding lemma, together with Lemma 3.1:

Lemma 5.2. *For an m-clause CNF, we have $\#\text{PrimeImplicants}_j(\varphi)\le 3^{n-j/k+1}$.*

Furthermore, we can repeat the analysis in the proof of Theorem 2 to get the following:

Theorem 4. *Let φ be an m-clause CNF, $k=\frac{2+\log_3(m)}{1-\log_3(2)}$, and $c=\frac{1-\log_3(2)}{\log_3(2)+1/k}$. For m-clause CNFs with $m\ge 3$,*

$$\#\text{PrimeImplicants}(\varphi)\le n\cdot 3^{n(1-c/k)+1}$$

In particular, for $k\ge 3$, we can take $c\approx .382$, and for $k\ge 10$, we can take $c\approx .505$.

Proof. We show that

$$\#\text{PrimeImplicants}_j(\varphi)\le 3^{n(1-c/k)+1}\text{ for }j\in[n].$$

We consider two cases. If $j\ge nc$, by Lemma 5.2,

$$\#\text{PrimeImplicants}_j(\varphi)\le 3^{n-j/k+1}\le 3^{n-nc/k+1}.$$

Otherwise, $j<nc$, and the analysis here is the same as before, since $3^{n(1-c/k)}\le 3^{n(1-c/k)+1}$. □

5.1.3 Obtaining Prime Implicants

Replacing the decoder DEC_π^k with DEC_π^m in the k-CNF construction gives immediately:

Theorem 5. *Let φ be an m-clause CNF. Let $k=\frac{2+\log_3(m)}{1-\log_3(2)}$, and $c=\frac{1-\log_3(2)}{\log_3(2)+1/k}$. We deterministically obtain the BCF in time*

$$O(|\varphi|\cdot n^{2\cdot k\cdot\log m+2}\cdot 3^{n(1-c/k)}).$$

5.2 Monotone k-CNFs

We get a tighter bound and a faster algorithm in the monotone case, as a result of the following fact:

Fact 5.3. *If φ is monotone, every prime implicant assigns every variable to either 1 or $*$.*

Proof. Let I be an implicant of φ. It suffices to show that if $\neg x_i\in I$ for some i, the assignment $I'=I\setminus\{\neg x_i\}$ is a smaller implicant of φ. Indeed, let J be some total assignment agreeing with I' on its fixed variables. If $J_i=0$, then J agrees with I well, and so satisfies φ. Otherwise, $J_i=1$, then $J\oplus i$ satisfies I, and hence φ, and $J\oplus i\le J$, so by monotonicity, J again satisfies φ, as required. □

It follows from this fact that in the monotone case, 0 never appears in any encoding of a prime implicant. We thus achieve a shorter encoding length, using the following analogue of Fact 3.1 shown by Paturi, Pudlák and Zane [PPZ97]:

Fact 5.4. *If ENC is a one-to-one, prefix-free encoding of S into strings formed from the alphabet $\{1, *\}$, and ENC has average code length ℓ, then $|S| \leq 2^{\ell}$.*

Applying this to the analysis of our encoding scheme, we get the following.

Corollary 5.5. *For a monotone k-CNF, the number of j-fixed prime implicants is at most $2^{n-j/k}$.*

Theorem 8. *If φ is a monotone k-CNF and $c \in (0, 1/2]$ is such that $H(c) \leq 1 - c/k$, then*

$$\#\text{PrimeImplicants}(\varphi) \leq n \cdot 2^{n(1-c/k)}.$$

In particular, for all $k \geq 2$, one can take $c = .282$.

Proof. Let $\#\text{PrimeImplicants}_j(\varphi)$ denote the number of j-fixed prime implicants. We'll show that $\#\text{PrimeImplicants}_j(\varphi) \leq 2^{n(1-c/k)}$ for each j. If $j \geq cn$, then Corollary 5.5 gives

$$\#\text{PrimeImplicants}_j(\varphi) \leq 2^{n-cn/k}.$$

Otherwise, $j < cn$. In this case we use the fact that there are at most $\binom{n}{j}$ prime implicants; prime implicants for monotone formulas contain only 1 and $*$, we only need to pick j out of n variables to assign 1. Using an entropy bound on the binomial coefficient,

$$\#\text{PrimeImplicants}_j(\varphi) \leq \binom{n}{j} \leq 2^{H(c)\cdot n},$$

where H is the binary entropy function. By our choice in c, $\#\text{PrimeImplicants}_j(\varphi) \leq 2^{n(1-c/k)}$. □

To show that the above bound limits the minimal DNF size for monotone k-CNFs, we rehearse an observation due to Quine [Qui54]:

Lemma 5.6. *If φ is a monotone function, then:*

1. *All prime implicants of φ are essential.*
2. *The Blake-Normal-Form of φ is minimal.*

Proof. Let φ be any monotone function. We show the first statement, as the second follows immediately. Let I be some prime implicant of φ viewed as a set of un-negated literals. Let σ be the total assignment that assigns everything in I to 1 and every other variable 0. We claim that no other prime implicant covers σ. Indeed, let J be a prime implicant that covers σ; we'll show that $J \subseteq I$. If not, there is some variable i such that $x_i \in J$, and $x_i \notin I$, but then J does not cover σ because σ is the ith variable assigned 0. Thus we have that $J \subseteq I$, as I was prime, it must be the case that $J = I$, which shows that the only prime implicant that covers σ is I. □

Denote by $\text{dnfsize}(\varphi)$ the size of the shortest DNF equivalent to φ. Combining 5.6 and 8 yields:

Corollary 5.7. *If φ is a monotone k-CNF, then $\text{dnfsize}(\varphi) \leq n \cdot 2^{n(1-c/k)}$, for $c = 0.282$.*

Thus when $k = o(n/\log(n))$, for large enough n the factor of n is negligible, and the above bound is tighter than the best-known bound of $2^{n(1-1/(100k))}$, shown by Miltersen, Radhakrishnan and Wegner [MRW05]. We note that one can also obtain Corollary 5.5 from the Satisfiability Coding Lemma [PPZ97], since every prime implicant is essential and so maps to some unique satisfying assignment, and the satisfying assignments can be partitioned by their j-sensitivity and then bounded using the lemma. However, unlike the Satisfiability Coding Lemma [PPZ97], our construction gives rise to an algorithm for obtaining the minimal DNF of a monotone k-CNF. Indeed, the construction used in the general k-CNF case and the analysis in Theorem 8 yield:

Theorem 7. *Let φ be a monotone k-CNF. Then for $c \in (0, 1/2]$ such that $H(c) = 1 - c/k$, we deterministically obtain the BCF (or, equivalently, the minimal DNF) in time*

$$O(|\varphi| \cdot n^{2k+1} \cdot 2^{n(1-c/k)}).$$

In particular, for all $k \geq 2$, one can take $c = .282$.

References

[CM78] Ashok K Chandra and George Markowsky. On the number of prime implicants. *Discrete Mathematics*, 24(1):7–11, 1978.

[DF59] B Dunham and R Fridshal. The problem of simplifying logical expressions. *The Journal of Symbolic Logic*, 24(1):17–19, 1959.

[DFLBM13] David Déharbe, Pascal Fontaine, Daniel Le Berre, and Bertrand Mazure. Computing prime implicants. In *2013 Formal Methods in Computer-Aided Design*, pages 46–52. IEEE, 2013.

[Hir17] Shuichi Hirahara. A duality between depth-three formulas and approximation by depth-two. *arXiv preprint arXiv:1705.03588*, 2017.

[JMSSS14] Said Jabbour, Joao Marques-Silva, Lakhdar Sais, and Yakoub Salhi. Enumerating prime implicants of propositional formulas in conjunctive normal form. In *European Workshop on Logics in Artificial Intelligence*, pages 152–165. Springer, 2014.

[McC56] Edward J McCluskey. Minimization of boolean functions. *The Bell System Technical Journal*, 35(6):1417–1444, 1956.

[MFSO97] Vasco M Manquinho, Paulo F Flores, João P Marques Silva, and Arlindo L Oliveira. Prime implicant computation using satisfiability algorithms. In *Proceedings Ninth IEEE International Conference on Tools with Artificial Intelligence*, pages 232–239. IEEE, 1997.

[MRW05] Peter Bro Miltersen, Jaikumar Radhakrishnan, and Ingo Wegener. On converting CNF to DNF. *Theoretical computer science*, 347(1-2):325–335, 2005.

[PPP99] Luigi Palopoli, Fiora Pirri, and Clara Pizzuti. Algorithms for selective enumeration of prime implicants. *Artificial Intelligence*, 111(1-2):41–72, 1999.

[PPZ97] Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 566–574. IEEE, 1997.

[Qui52] Willard V Quine. The problem of simplifying truth functions. *The American mathematical monthly*, 59(8):521–531, 1952.

[Qui54] Willar V Quine. Two theorems about truth-functions. 1954.

[Qui55] Willard V Quine. A way to simplify truth functions. *The American mathematical monthly*, 62(9):627–631, 1955.

[SU02] Marcus Schaefer and Christopher Umans. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT news*, 33(3):32–49, 2002.

[Tal14] Navid Talebanfard. *On the Combinatorics of SAT and the Complexity of Planar Problems*. PhD thesis, Department Office Computer Science, Aarhus University, 2014.

[Tal16] Navid Talebanfard. On the structure and the number of prime implicants of 2-CNFs. *Discrete Applied Mathematics*, 200:1–4, 2016.