The complexity of SPP formula minimization

David Buchfuhrer[*]
Computer Science Department
California Institute of Technology
Pasadena, CA 91125
dave@cs.caltech.edu

Abstract

Circuit minimization is a useful procedure in the field of logic synthesis. Recently, it was proven that the minimization of $(\vee, \wedge, \neg)$ formulae is hard for the second level of the polynomial hierarchy [BU08]. The complexity of minimizing more specialized formula models was left open, however. One model used in logic synthesis is a three-level model in which the third level is composed of parity gates, called SPPs. SPPs allow for small representations of Boolean functions and have efficient heuristics for minimization. However, little was known about the complexity of SPP minimization. Here, we show that SPP minimization is complete for the second level of the Polynomial Hierarchy under Turing reductions.

[*]Supported by NSF CCF-0346991, CCF-0830787 and BSF 2004329.

1 Introduction

Circuit minimization problems are an interesting class of problems contained in the second level of the Polynomial-Time Hierarchy (PH). Given a circuit C computing a function f, the goal is to find the smallest circuit C' also computing f. The complexity of circuit minimization depends heavily upon the model of circuit being minimized as well as how the size of a circuit is calculated. Here we use boolean formulae as our model. The details of this model and the size function are defined later.

As a simple example, take the formula $a \rightarrow b$. It could be expressed either by the formula

$$(a \wedge b) \vee \neg a$$

or by the simpler and more common

$$\neg a \vee b.$$

Since we are measuring the size by the number of occurrences of variables, the size of the first formula is 3, while the size of the second is 2. Even with this simple example, the fact that the first formula is not minimal is not immediately clear without the knowledge that it computes $a \rightarrow b$. With larger, more complicated formulae, the problem becomes even more difficult.

Circuit minimization has practical applications in fields such as logic synthesis and hardware design. Two popular variants of circuit minimization are unbounded-depth and constant depth formula minimization over $(\wedge, \vee, \neg)$ formulae. The 2-level (DNF) version was proven Σ_2^P complete in [Uma98]. More recently, the constant depth version for all constants greater than 2 as well as the unlimited depth version were proven to be hard for the second level of the PH under Turing reductions in [BU08]. More recently, three-level formula minimization in which the first level is an OR gate, the second level is AND gates, and the third level is composed of XOR gates, has been introduced [LP99]. Such a formula is referred to as a Sum of Pseudoproducts (SPP) in the literature.

Definition 1.1 (SPP Formulae)**.** *An SPP formula is the disjunction of the conjunction of parity formulae. In other words, an SPP formula contains a single OR gate of unlimited fan-out, to which the inputs are unlimited fan-out AND gates which in turn take unlimited fan-out XOR gates as input. The size of an SPP formula is the number of times the input variables appear in it.*

Rather than allow negations of input variables, we allow the XOR gates to take constants as inputs. Adding a true input is equivalent to negating one of the input variables and does not increase the formula size. The number of true inputs thus determines whether an XOR gate computes odd or even parity on the input variables.

Figure 1 contains an example of an SPP formula computing the high bit of a 2-bit adder. The inputs are x_1x_0 and y_1y_0 and the nodes marked $\bigoplus$ are XOR gates. The sub-formula computed by an AND gate is referred to as a pseudoproduct.

Definition 1.2 (Pseudoproduct)**.** *A pseudoproduct is the conjunction of XORs.*

Note that when we refer to XORs above, we mean the function computed by an XOR gate rather than the gate itself. Both meanings are used throughout this paper.

SPP formula minimization has been a recent subject of study in the field of logic synthesis [BCDV08, CB02, Cir03]. Some research has found inefficient but exact minimization algorithms, while other research is directed toward finding more efficient heuristics. Little has been known about the formal complexity of SPP minimization. Characterizing the complexity of SPP minimization is important, as SPPs are well-suited for use in practical situations such as representation of arithmetic functions [LP99, JSA97].

Note that SPP formulae are a generalization of DNF formulae. Recall that a DNF formula is simply the disjunction of several terms, each of which is the conjunction of some of the input

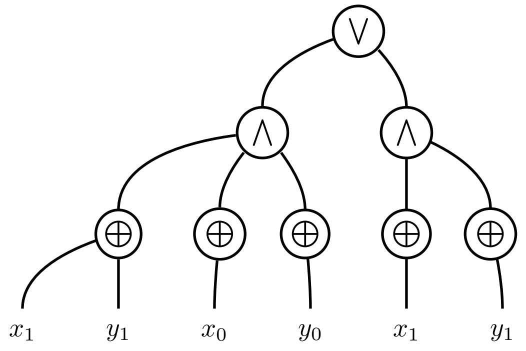

Figure 1: An SPP formula for the high bit output of a 2-bit adder.

variables and their negations. One reason that SPP formulae are useful in logic synthesis is that they allow for much smaller representations of many Boolean functions than DNF formulae. The simplest such example is that of the parity function. Since SPP formulae contain XOR gates of unlimited fan-out, parity only requires a single XOR gate with a linear number of inputs. By contrast, a CNF or DNF for parity requires exponential size.

We formally define the SPP minimization problem as

Problem 1.1 (Minimum SPP Formula (MSF))**.** *Given an SPP formula S and an integer k, is there an SPP formula of size at most k which is equivalent to S?*

Here, we show that the minimization of SPP formulae is hard for the second level of the PH under Turing reductions. This result provides the theoretical backing for work on inefficient exact minimization algorithms [Cir03] and heuristics [BCDV08]. It also closes a gap in our knowledge of the complexity of circuit minimization. Σ_2^P hardness demonstrates that polynomial time algorithms making use of NP or coNP queries cannot exist unless the PH collapses. This is an important result to those working in logic synthesis, as queries to problems such as the coNP-hard tautology problem are often used in the design of algorithms and heuristics.

1.1 Description of the reduction

This section contains a high-level description of the reduction used to show that Problem 1.1 is Σ_2^P-complete under Turing reductions, in order to facilitate understanding of a more technical description later. We also compare the reduction in this paper to a related reduction in [BU08].

The problem we reduce from is Modified Succinct Set Cover, which was shown to be Σ_2^P-complete in [BU08].

Problem 1.2 (modified succinct set cover (MSSC))**.** *Given a DNF formula D on variables*

$$v_1, v_2, \ldots, v_m, x_1, x_2, \ldots, x_n$$

and an integer k, is there a subset $I \subseteq \{1, 2, \ldots n\}$ with $|I| \leq k$ and for which

$$D \vee \bigvee_{i \in I} \neg x_i \equiv \left(\bigvee_{i=1}^{m} \neg v_i \vee \bigvee_{i=1}^{n} \neg x_i \right)?$$

Less formally, the goal is to determine whether there exists a small subset I of the x_i variables, for which $\bigvee_{i \in I} x_i$ accepts every assignment not accepted by D, except for the all true assignment. In this way, MSSC can be seen as a succinct form of the set cover problem, in which the points accepted by D and the formulae $\neg x_i$ are the sets which form the instance.

In order to reduce MSSC to MSF, we attempt to split the SPP formula of the MSF instance into one portion which computes D and another which accepts $\bigvee_i \neg x_i$. To accomplish this, we add a variable z, the value of which determines whether the formula computes D on the rest of the variables, or $D \vee \bigvee_i \neg x_i$. With this variable added, the formula should look like $D \vee (z \wedge \bigvee_i \neg x_i)$. Because SPP formulae only have one OR gate at the top level, the particular form is more similar to $D \vee \bigvee_i (z \wedge \neg x_i)$.

The difficult direction of the reduction is showing that if the MSSC instance is negative, so is the MSF instance. In order to prevent a small equivalent SPP formula from existing if the MSSC is negative, we attach different weights to the variables. Rather than simply counting each occurrence of a variable as 1 toward the size of the formula, we wish to count each variable v as $w(v)$ to add flexibility to our reduction.

In order to increase the size contribution of each variable, we replace each occurrence of each variable v with $w(v)$ new variables, so they contribute $w(v)$ toward the size at each occurrence of v. This is achieved exactly by replacing each variable v with $v_1 \oplus v_2 \oplus \cdots \oplus v_{w(v)}$. The fact that this transformation preserves minimum formula size is shown by Lemma 2.3.

We weight the z variable in a less general way that is better suited for achieving the desired form. We replace z by $z_1 \wedge \cdots \wedge z_\ell$, where ℓ is an integer value particular to the instance of MSSC. In order to avoid confusion, we do not refer to this as weighting in the more technical description of the reduction, and simply state the reduction explicitly with variables $z_1, \ldots, z_\ell$.

The reduction approach we use in this paper is superficially similar to one contained in [BU08], which proves that minimization of $(\vee, \wedge, \neg)$ formulae of both fixed depth ≥ 3 and unrestricted depth are Σ_2^P-complete under Turing reductions. Consider the depth-3 version, which is most similar to SPP minimization.

Problem 1.3 (MINIMUM EQUIVALENT DEPTH 3 EXPRESSION (MEE_3))**.** *Given a depth* 3 *Boolean* $(\vee, \wedge, \neg)$ *formula F and an integer k, is there an equivalent depth* 3 *formula of size at most k?*

Here, the depth ignores NOT gates. The reduction in [BU08] showing that MEE_3 is Σ_2^P-complete under Turing reductions also reduced from MSSC, and used the same basic strategies. The original variables were given different weights to avoid the formula being too small when the initial MSSC instance is negative. Extra variables were added which split the formula into a portion computing D and a portion computing $D \vee \bigvee_i \neg x_i$, playing a similar role to the z_i variables in this paper, but with an altogether different form. In the SPP reduction shown in this paper, we create a formula with a form similar to

$$D \vee \bigvee_i (z \wedge \neg x_i).$$

By contrast, in [BU08] the formula resulting from the reduction is more similar to

$$D \vee \left(z \wedge \bigvee_i \neg x_i \right).$$

The two above forms are different ways of expressing the same formula. The big difference, however, is the number of occurrences of the z variable. In [BU08], the z variable is given such a heavy weight that it must appear only once in a minimal formula. The entire proof depends on this fact. This is not possible with an SPP formula, however, as the z would then appear in only one pseudoproduct, and this pseudoproduct would thus be responsible for computing $\bigvee_i \neg x_i$. Lemma 1.1 demonstrates why this prevents any such reduction from working.

Lemma 1.1. *No pseudoproduct can compute $\neg a \vee \neg b$.*

Proof. Suppose by way of contradiction that some pseudoproduct P computed $\neg a \vee \neg b$. Consider the XORs contained in P. If P contains an XOR of only a single variable, we assume without loss of generality it computes either a or $\neg a$. Clearly, if the XOR computes a, P does not accept $a =$ false,$b =$ false, and thus cannot compute $\neg a \vee \neg b$. Similarly, if P contains a pseudoproduct computing $\neg a$, P does not accept $a =$ true, $b =$ false and again cannot compute $\neg a \vee \neg b$.

By elimination, P $\neg a \vee \neg b$ can only contain an XOR of both a and b. Such an XOR cannot accept both $a =$ false, $b =$ false and $a =$ true, $b =$ false, as these assignments differ in parity. Since P computes $\neg a \vee \neg b$, it must accept both of these assignments. Thus, P cannot contain only an XOR of both a and b.

The only remaining option is that P contains no XORs, and is thus trivial. Since P computes the non-trivial function $\neg a \vee \neg b$, we have a contradiction, proving the lemma. □

Lemma 1.1 is important because it demonstrates that a single pseudoproduct is not powerful enough to accept formulae like

$$\bigvee_i (z \wedge \neg x_i),$$

which can be restricted to $\neg x_i \vee \neg x_j$ by restricting z to true and all other variables except x_i and x_j to false.

Thus, we require several copies of the z variable in any reduction from MSSC to MSF using the general approach developed in [BU08]. In a depth-3 Boolean formula like those under consideration in the [BU08] reduction from MSSC to MEE$_3$, the sub-formula containing the z variable is a DNF, and is therefore not limited in computational capability like a pseudoproduct.

Since the z variable cannot occur only once, the relative weights must be completely different from those used in [BU08], in order to allow for an unknown number of z variables. Rather than a single z variable carrying more weight than the rest of the formula combined, all the z variables combined carry less weight than any other single variable. This is a huge change, and just the existence of multiple copies of the z variables alone completely destroys many of the proof techniques used in [BU08]. We are able to get a handle on the structure of the pseudoproducts containing the z variables by developing new proof techniques. One key structural lemma used in obtaining our results is Lemma 1.2.

Lemma 1.2. *If a pseudoproduct computes the conjunction of n variables $z_1, \ldots, z_n$, the pseudoproduct contains at least n XORs.*

The proof of Lemma 1.2 is contained in Section 2.3 and utilizes simple linear algebra.

1.2 Outline

Section 2 contains all of our results. Section 2.1 contains some background results that carry over from complexity results for DNF formulae. Section 2.2 contains an explanation of weighting necessary to the main reduction. Section 2.3 contains the proof that Problem 1.1 is Σ_2^P-complete, as outlined in Section 1.1.

Finally, Section 3 contains a summary of the results as well as a discussion of related problems that remain open.

2 Results

In this section, we present new classifications for the complexity of problems related to SPP formulae. We begin in Section 2.1 with a few simple results, then continue in Section 2.3 with the proof of the Σ_2^P completeness of MSF under Turing reductions.

2.1 Complexity of Related Problems

In this section, we note some complexity results that remain unchanged from the corresponding results for related DNF problems.

Problem 2.1 (SPP Equivalence)**.** *Given two SPP formula S, T, do both S and T compute the same function?*

Proposition 1. *SPP Equivalence is coNP-complete.*

While the proof is quite simple, this result is important to note, as not all types of logic formulae are known to share this result. For example, the equivalence of two ordered binary decision diagrams (OBDDs) is decidable in P [Bry86].

Proof. It is a well-known result that deciding whether two DNF formulae S, T compute the same formula is coNP-complete. Since DNF formulae are a special case of SPP formulae, the result that SPP Equivalence is coNP-complete immediately follows. □

Problem 2.2 (DNF Irredundancy)**.** *Given a DNF formula T and an integer k, does there exist an equivalent formula T' consisting of the disjunction of at most k terms from T?*

The irredundancy problem is important because it is often used in heuristics for DNF minimization problems, despite having been proven Σ_2^P hard to even approximate in [Uma99]. We see here that the corresponding problem for SPP formulae shares the same complexity classification.

Problem 2.3 (SPP Irredundancy)**.** *Given an SPP S and an integer k, is there an equivalent SPP S' composed of the disjunction of k pseudoproducts from S?*

Theorem 2.1. *SPP Irredundancy is Σ_2^P hard to approximate to within a factor of n^ϵ for some $\epsilon > 0$, where n is the input size.*

Proof. As shown in [Uma99], DNF Irredundancy is Σ_2^P-hard to approximate within n^ϵ for some $\epsilon > 0$. As in the proof of Proposition 1, we note that DNF formulae are a special case of SPP formulae. In particular, the terms of a DNF are a special case of the pseudoproducts of an SPP, so SPP Irredundancy is also Σ_2^P-hard to approximate within n^ϵ. □

Prime implicants are also important to DNF minimization, as a minimum DNF is composed only of prime implicants.

Definition 2.1 (Prime Implicant)**.** *A prime implicant of a DNF D is a term T which implies D and does not imply any other term which implies D.*

An implicant $x_1 \wedge \cdots \wedge x_n$ of D is prime iff there is no k such that $x_1 \wedge \cdots \wedge x_{k-1} \wedge x_{k+1} \wedge \cdots \wedge x_n$ is also an implicant of D. Since prime implicants are necessary for DNF minimization, it is important to be able to determine whether a term is a prime implicant.

Problem 2.4 (IS-PRIMI)**.** *Given a DNF D and a term T, is T a prime implicant of D?*

IS-PRIMI was proven DP-complete independently in both [GHM08] and [UVSV06].

Prime pseudoproducts are a generalization of prime implicants to SPP formulae.

Definition 2.2 (Prime Pseudoproduct)**.** *A prime pseudoproduct of an SPP S is a pseudoproduct P which implies S, but does not imply any other pseudoproduct which implies S.*

Prime pseudoproducts characterize SPP with a minimum number of XOR gates [LP99] rather than minimum SPP formulae as defined in this paper. Still, they are important to the study of SPP formulae, so we consider the complexity of determining whether a pseudoproduct is prime.

Problem 2.5 (SPP Prime Pseudoproduct (SPP-PP))**.** *Given an SPP S and a pseudoproduct P, is P a prime pseudoproduct of S?*

We will show that SPP-PP is DP-hard.

Definition 2.3 (DP)**.** *The class DP consists of the set of languages L such that there exist languages $L_1 \in \mathrm{NP}$ and $L_2 \in \mathrm{coNP}$ such that $L = L_1 \cap L_2$.*

Note that DP $\supseteq$ NP $\cup$ coNP, since L_1 and L_2 can be different languages in the above definition. The reduction we will use is the same reduction used in [GHM08] to prove that IS-PRIMI is DP-hard. The reduction is from SAT-UNSAT.

Problem 2.6 (SAT-UNSAT)**.** *Given CNF formulae α, β, is α satisfiable and β unsatisfiable?*

Theorem 2.2. *SPP-PP is DP-hard.*

Proof. We use the same reduction used in [GHM08] to show IS-PRIMI is DP-hard. Given an instance $\langle \alpha, \beta \rangle$ of SAT-UNSAT, we create the SPP-PP instance $\langle \neg\alpha \vee (\neg\beta \wedge y), y \rangle$, where y is a new variable which does not appear in α or β. Since this reduction gives a valid instance of IS-PRIMI, it also gives a valid instance of SPP-PP. This follows because a DNF is a special case of an SPP and y happens to be a pseudoproduct.

Now, we need only see that y is a prime pseudoproduct of f iff y is a prime implicant of f to see that the reduction works for the SPP case as well. First, it is clear that whether y implies f does not depend on whether we represent f by a DNF or an SPP. We now consider which pseudoproducts are implied by y. Any such pseudoproduct must contain only XORs which are implied by y. Such an XOR cannot contain any other variable, as y alone would not imply such an XOR.

Thus, a pseudoproduct implied by y contains only XORs which contain at most the single variable y, so y only implies itself and the trivially true pseudoproduct, λ. Since these are both DNF terms as well as pseudoproducts, y implies a DNF term implicant of f iff y implies a pseudoproduct implicant of f. So by the definitions of prime pseudoproduct and prime implicant, y is a prime pseudoproduct of f iff y is a prime implicant of f, regardless of the function under consideration. Setting $f = \neg\alpha \vee (\neg\beta \wedge y)$, we see that SPP-PP is DP-hard as well as showing that IS-PRIMI is DP-hard. □

Note that we only achieve a hardness result here and not a completeness result. One key feature of IS-PRIMI that is used to prove containment in DP is that in order to check that $a_1 \wedge \cdots \wedge a_n$ is prime, we need only verify that the n terms obtained by removing one of the a_is do not imply f. With pseudoproducts, it is unclear how to check primality in NP. Thus, we only have a lower-bound complexity result.

In this section, we have seen that complexity results for equivalence and irredundancy of SPP formulae can be inferred directly from the corresponding results for DNF formulae because DNF formulae are a special case of SPP formulae. These results were simple because the problems only depend on the formulae taken as inputs, and do not depend on properties of any other formula. This preserves the DNF versions as special cases of the SPP versions. In general, formula minimization problems differ in that the problem instance is compared to all formulae of size less than k, necessitating a different reduction when the type of formulae under consideration is changed. So the result that DNF formula minimization is Σ_2^P-complete [Uma01] does not immediately carry over to SPP formula minimization.

The Σ_2^P-completeness result does not directly carry over from DNFs to SPPs, but one might hope that the proof techniques used would, as happened with the DP-hardness proofs of IS-PRIMI and SPP-PP. However, the proof of DP-hardness for prime implicants had the convenient feature that the DNF term y is a prime implicant iff it is a prime pseudoproduct. The proof of DNF minimization being Σ_2^P-complete does not have the analogous feature that the DNF produced is a minimum DNF iff it is a minimum SPP. So a new reduction is necessary.

2.2 Weighted variables

Normally, we measure the size of an SPP formula by the number of occurrences of variables within it. Each occurrence counts as 1 toward the total size. Another notion of size might be to assign a positive integer weight $w(v)$ to each variable v, and count each occurrence of the variable v as $w(v)$ toward the total size. We will show a transformation from positive integer weights to the normal size measure in which each variable counts as 1, which preserves minimum formula size.

Since the SPP formulae in consideration have unlimited fanout XOR gates, we can replace a variable v of weight $w(v)$ with the XOR of $w(v)$ new variables, $v_1 \oplus \cdots \oplus v_{w(v)}$ in order to eliminate the need for directly assigning weights to variables. This weighting is achieved by adding the new v_i variables to each XOR gate containing v, and removing v completely.

Given an SPP formula F with associated weight function $w(v)$, we call F the *weighted* version of the formula. Let F' be the formula obtained by replacing every variable v with the XOR of $w(v)$ new variables. We call F' the *expanded* version of the formula. Note that if F computes f and F' computes f',

$$f(x_1 \oplus \cdots \oplus x_{w(x)}, y_1 \oplus \cdots \oplus y_{w(y)}, \ldots)$$

is equivalent to

$$f'(x_1, x_2, \ldots, x_{w(x)}, y_1, y_2, \ldots, y_{w(y)}, \ldots).$$

We denote the weighted size of F with weights $w(v)$ by $|F|_w$, and we denote the size of F' by $|F'|$.

As a quick example, consider the formula $\neg x \oplus y$. Letting $w(x) = 2$ and $w(y) = 3$, we arrive at the expanded form shown in Figure 2. Recall that we represent $\neg x \oplus y$ by $x \oplus y \oplus 1$ in this paper.

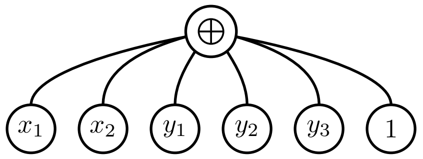

Figure 2: The expanded form of $\neg x \oplus y$ with weights $w(x) = 2, w(y) = 3$

The expansion described above differs from the expansion strategy used in [BU08], in which a conjunction of $w(v)$ new variables was used rather than an XOR. Our strategy has the advantages that it handles negation without requiring a possible increase in the depth of the formula and can be easily applied to SPP formulae. Lemma 2.3 demonstrates that applying this simple transformation is equivalent to positive integer weighting. Before we get to Lemma 2.3, we define the weighted version of MSF.

Problem 2.7 (Weighted MSF (W-MSF)). *Given an SPP formula S, a list of weights w for each variable in unary, and an integer k, is there a formula S' equivalent to S for which $|S'|_w \leq k$?*

We refer to a formula which is minimum with respect to w as a w-minimum formula. In order to prove that MSF and W-MSF are equivalent, we need to define the notion of a restriction.

Definition 2.4 (Restriction). *A restriction of a function f to a partial assignment σ is simply the remaining function on all other variables when the variables contained in σ are fixed according to σ. Similarly, a restriction of a formula F to σ results in the formula F with occurrences of variables in σ replaced with values according to σ.*

A restriction will reduce the size of the formula, as constants do not count toward formula size. Note that if we restrict an SPP formula, the resulting formula is also an SPP formula.

Lemma 2.3. *Let f be a Boolean function and w be a positive integer weighting function for the variables of f. If F is a w-minimum formula for f and F' is a minimum formula equivalent to the expanded version of F, then $|F|_w = |F'|$.*

Proof. Clearly, $|F'| \leq |F|_w$, as the expanded version of F has size exactly $|F|_w$.

Now we will see that $|F'| \geq |F|_w$. For each variable v in F which has been replaced by variables $v_1, v_2, \ldots, v_{w(v)}$ in F', find the index i^* such that v_{i^*} occurs the least often in F' of all the variables $v_1, v_2, \ldots, v_{w(v)}$. Let F^* be the restriction of F' for which $v_i = \text{false}\ \forall i \neq i^*$, for each variable v. Note that F^* is equivalent to F, as

$$\text{false} \oplus \cdots \oplus \text{false} \oplus v_{i^*} \oplus \text{false} \oplus \cdots \oplus \text{false} = v_{i^*}.$$

We now compare F^* to F' one variable at a time. Suppose v_{i^*} occurs α times in F'. Note that v_{i^*} also occurs α times in F^*. The variables $v_1, v_2, \ldots, v_{w(v)}$ contribute at least $w(v)\alpha$ to $|F'|$, since i^* was chosen such that v_{i^*} occurs least frequently of all the v_i variables in F', and there are $w(v)$ such variables. Since v_{i^*} contributes exactly $w(v)\alpha$ to $|F^*|_w$, we see that $|F'| \geq |F^*|_w$. Thus, since F is a w-minimum formula equivalent to F^*, we have $|F'| \geq |F^*|_w \geq |F|_w$.

Since $|F'| \leq |F|_w$ and $|F'| \geq |F|_w$, we have $|F'| = |F|_w$. □

Now, we apply Lemma 2.3 to show that W-MSF and MSF are poly-time equivalent.

Theorem 2.4. *There exist polynomial-time reductions between W-MSF and MSF.*

Proof. First, we will see a reduction from MSF to W-MSF. Take an instance $\langle S, k\rangle$ of MSF and produce the instance $\langle S, w, k\rangle$ of W-MSF in which $w(v) = 1\ \forall v$. Clearly, there exists an formula S' equivalent to S of size at most k iff there exists a formula S' equivalent to S for which $|S'|_w \leq k$, since $|S'|_w$ is equal to the unweighted size when all weights are 1. This reduction only takes linear time, as we need only write down the MSF instance and a constant weight function.

Now we reduce W-MSF to MSF. Since the weights are written in unary, they are all linear in the instance size. Thus, we can convert $\langle S, w, k\rangle$ to $\langle S', k\rangle$ by setting S' to be the expanded version of S. Since this only requires replacing each occurrence of each variable with a linear number of new variables, we have a polynomial-time reduction. Furthermore, by Lemma 2.3, the w-minimum formula equivalent to S is of size at most k iff the minimum formula equivalent to S' is of size at most k, proving the theorem. □

Since these two problems are polynomial-time equivalent, we only need to prove that one of them is Σ_2^P complete under Turing reductions to see that they both are. We will show that MSF is Σ_2^P complete under Turing reductions via W-MSF.

2.3 Main Result

In this section, we show that Problem 1.1, the SPP minimization problem, is Σ_2^P complete under Turing reductions by reducing from MSSC to W-MSF. The following steps describe how a Turing machine with access to an oracle for W-MSF can solve MSSC.

- We are given an instance $\langle D, x_1, x_2, \ldots, x_n, k\rangle$ of MSSC, where D is a DNF formula over variables $v_1, \ldots, v_m, x_1, \ldots, x_n$.
- Set $\ell = 2k(k+1)(|D|+1)$.
- Create new variables $z_1, \ldots, z_\ell$.
- Set the weight function w such that $w(v_i) = 2k^2\ell$, $w(x_i) = k\ell$ and $w(z_i) = 1$ for all i.
- Using binary search, find the size of the w-minimum SPP formula for D using $O(\log |D|_w)$ oracle calls. Call this size τ.

- Set $S = D \vee (Z \wedge \neg x_1) \vee (Z \wedge \neg x_2) \vee \cdots \vee (Z \wedge \neg x_n)$ where $Z = z_1 \wedge z_2 \wedge \cdots \wedge z_\ell$.
- Finally, we make one last oracle call to determine whether there is an SPP formula S' equivalent to S for which $|S'|_w \leq \tau + k(k+1)\ell$. Accept iff such an S' exists.

Remark 1. *Since this reduction only uses logarithmically many adaptive oracle calls, standard techniques can be used to turn this into a non-adaptive reduction with polynomially many oracle calls.*

We begin with a proof of Lemma 1.2.

Proof (of Lemma 1.2). Suppose by way of contradiction that the conjunction of n variables can be computed by a pseudoproduct P with $n' < n$ XOR gates. Associate with each variable z_i the vector $\omega^{(i)} \in \mathbb{Z}_2^{n'}$ in which the jth position of $\omega^{(i)}$, $\omega_j^{(i)}$, is equal to 1 iff z_i is contained in the jth XOR.

Now we can match each assignment σ of truth values to the z_i variables to the sum $W(\sigma) = \sum_i z_i \omega^{(i)}$. Note that the jth position of $W(\sigma)$, $W(\sigma)_j$, is equal to $\sum_i z_i \omega_j^{(i)}$, or simply the parity of the number of true variables contained in the jth XOR. In order for an XOR to be true, the parity of the true variables it contains plus the parity of the true constants it contains must be odd.

Thus, for P to evaluate to true on assignment σ, $W(\sigma)_j$ should be equal to 1 iff the jth XOR contains an even number of true constants, in order for the total parity of the variables and constants to be odd. Let W^* be the vector for which $W^*_j = 1$ iff the jth XOR contains an even number of true constants. P evaluates to true on assignment z iff $W(\sigma) = W^*$.

Since P computes a satisfiable function, there is some assignment σ such that $W(\sigma) = W^*$. However, since $\omega^{(1)}, \ldots, \omega^{(n)}$ is a collection of n vectors over the vector space $\mathbb{Z}_2^{n'}$ of dimension $n' < n$, they are not linearly independant. Thus, there are at least 2 linear combinations of $w_1, \ldots, w_n$ equal to any vector in the span of $\omega^{(1)}, \ldots, \omega^{(n)}$, including W^*. Therefore, P accepts at least two assignments, which contradicts the fact that P computes $z_1 \wedge z_2 \wedge \cdots \wedge z_n$, which only accepts one assignment. □

Another helpful lemma was proven in [BU08]:

Lemma 2.5. *Let $t_1, t_2, \ldots, t_n$ be a set of variables, and S a set of assignments of true/false values to $t_1, \ldots, t_n$. A minimum formula accepting at least S and not the all-true assignment to variables $t_1, t_2, \ldots, t_n$ is of the form $\bigvee_{i \in I} \neg t_i$ for some $I \subseteq \{1, 2, \ldots, n\}$.*

Although this lemma was only proven for $(\vee, \wedge, \neg)$-formula in [BU08], it holds true for weighted SPP as well using the same proof. The proof in both cases relies on transforming a formula F accepting at least S but not the all-true assignment into the form $\bigvee_{i \in I} \neg t_i$, by setting I as the set of all variables mentioned in F. Everything in S will still be accepted, the all-true assignment will still be rejected, and the size is not increased.

One final necessary lemma will allow us to determine the placement of the z_i variables in S' when $|S'| \leq \tau + k(k+1)\ell$.

Lemma 2.6. *Let S' be an equivalent SPP formula to the SPP formula S created by the Turing reduction described at the beginning of Section 2.3. Let U be the subset of pseudoproducts in S' that accept an assignment not accepted by D. Let H_P be the set of z_i variables in pseudoproduct P which occur in some XOR in P that contains only constants and z_i variables. Either $|S'|_w > \tau + k(k+1)\ell$ or there exists some index i^* such that $z_{i^*} \in H_P\ \forall P \in U$.*

Proof. Suppose that there does not exist an index i^* such that $z_{i^*} \in H_P\ \forall P \in U$.

Consider a pseudoproduct $P \in U$. Let ${H_P}^C$ denote the complement of H_P within the set of z_i variables. So ${H_P}^C$ is the set of z_i variables that never appear in an XOR in P consisting of only z_i variables and constants. Since $P \in U$, there exists some assignment σ accepted by P but not by D.

Let P' be the restriction of P to σ on all variables which are not members of ${H_P}^C$. Thus, P' computes some function on the variables in ${H_P}^C$. Recall that $S = D \vee \bigvee_i (z_1 \wedge \cdots \wedge z_\ell \wedge \neg x_i)$. Since σ is not accepted by D, S becomes $\bigwedge_{z_i \in {H_P}^C} z_i$ under the restriction to σ on all variables outside of ${H_P}^C$.

P' cannot accept anything not accepted by S', so P' can only accept when $\bigwedge_{z_i \in {H_P}^C} z_i$ is true. Furthermore, it must accept in this case, as P accepted σ. So P' computes exactly $\bigwedge_{z_i \in {H_P}^C} z_i$. Thus, by Lemma 1.2, there are at least $|{H_P}^C|$ XORs in P'. Note that this does not include XORs which contain only constants. These can be safely ignored, as they must all reduce to true since P' does not compute a constant function.

Each of the $|{H_P}^C|$ non-trivial XORs in P' corresponds to an XOR in P that contains a variable other than one of the z_is. This follows from the definition of ${H_P}^C$ and the fact that each non-trivial XOR in P' contains a member of ${H_P}^C$. Since all variables other than the z_i are weighted at least ℓ, $|P|_w \geq \ell |{H_P}^C|$. Furthermore, since no z_i is contained in H_P $\forall P \in U$,

$$\bigcup_{P \in U} {H_P}^C = \{z_1, \ldots, z_\ell\}.$$

So

$$\sum_{P \in U} |{H_P}^C| \ell \geq \ell^2,$$

which is important because $|S'|_w \geq \sum_{P \in U} |P|_w \geq \sum_{P \in U} |{H_P}^C| \ell$. Thus, we have

$$\begin{aligned}
|S'|_w &\geq \ell^2 \\
&= 2k(k+1)(|D|+1)\ell \\
&= 2k(k+1)\ell|D| + 2k(k+1)\ell \\
&> \tau + 2k(k+1)\ell \\
&> \tau + k(k+1)\ell,
\end{aligned}$$

where we know $2k(k+1)\ell|D| > \tau$ since $w(v) \leq 2k^2\ell < 2k(k+1)\ell$, so $\tau \leq |D|_w < 2k(k+1)\ell|D|$ for every variable v.

Thus, either there is some index i^* such that $z_{i^*} \in H_P$ $\forall P \in U$ or $|S'| > \tau + k(k+1)\ell$. $\square$

The main result that W-MSF is Σ_2^P-complete under Turing reductions, and therefore so is MSF, follows from Theorem 2.7.

Theorem 2.7. *Let S be the SPP formula resulting from the Turing reduction at the beginning of Section 2.3. The W-MSF instance $\langle S, w, \tau + k(k+1)\ell \rangle$ is positive iff the original* MSSC *instance was positive as well.*

Proof. First, we will show that a positive MSSC instance implies a positive W-MSF instance. Let $I \subseteq [n]$ such that $|I| \leq k$ and $D \vee \bigvee_{i \in I} \neg x_i \equiv \bigvee_{i=1}^m \neg v_i \vee \bigvee_{i=1}^n \neg x_i$. Since the MSSC instance is positive, such an I exists. Let $\widehat{D}$ be a w-minimum formula equivalent to D. Note $|\widehat{D}|_w = \tau$. We construct a formula

$$S' = \widehat{D} \vee \left[\bigvee_{i \in I} \left(\neg x_i \wedge \bigwedge_{j=1}^{\ell} z_j \right) \right].$$

which is equivalent to S since when we restrict such that $\bigwedge_{j=1}^{\ell}$ is false, it computes D and when we restrict such that $\bigwedge_{j=1}^{\ell}$ is true, it computes $D \vee \bigvee_{i \in I} \neg x_i \equiv \bigvee_{i=1}^m \neg v_i \vee \bigvee_{i=1}^n \neg x_i$. Thus, S' and S compute the same functions under any restriction to the z_i variables, so $S' \equiv S$. Note that $|S'|_w = |\widehat{D}|_w + \sum_{i \in I}(w(x_i) + \ell) = \tau + k(k\ell + \ell) = \tau + k(k+1)\ell$. Thus, a positive instance of MSSC implies a positive instance of W-MSF.

Now we prove that a negative instance of MSSC implies a negative instance of W-MSF. We will show that if the MSSC instance is negative, so is the W-MSF instance by first restricting to the portion of the formula computing D via the upcoming Claim 2.7.1 to achieve a τ lower bound. We then show that the rest of S' is of size greater than $k(k+1)\ell$ by the upcoming Claim 2.7.2.

Suppose that the instance of MSSC is negative. Assume by way of contradiction that the W-MSF instance is positive. By this assumption, there exists some formula S' which is equivalent to S and for which $|S'|_w \leq \tau + k(k+1)\ell$. Let U be the subset of pseudoproducts in S' that accept an assignment to the input variables not accepted by D. Let H_P be defined as in Lemma 2.6 to be the set of z_i variables that occur in some XOR in a pseudoproduct P containing only z_i variables and constants. By Lemma 2.6, there is some index i^* such that $z_{i^*} \in H_P\ \forall P \in U$, since $|S'|_w \leq \tau + k(k+1)\ell$.

Claim 2.7.1. *Every member of U becomes false under the restriction $z_i = \text{true}\ \forall i \neq i^*$ and $z_{i^*} = \text{false}$.*

Proof. Let $P \in U$. Since $P \in U$, P accepts some assignment σ not accepted by D. Since σ is accepted by S' but not by D, all z_i are true in σ. This follows from S' being equivalent to $S = D \vee \bigvee_i (z_1 \wedge \cdots \wedge z_\ell \wedge \neg x_i)$. Consider the assignment σ' in which all variables are set according to σ, except for z_{i^*}, which is set to false, (so σ and σ' differ only on the assignment to z_{i^*}.)

Since $z_{i^*} \in H_P$, some XOR gate X of P contains z_{i^*}, and only contains z_i variables and constants. Since P accepts σ, so must X. Since σ and σ' differ only on z_{i^*} and X accepts σ, X cannot accept σ', as the differing value of z_{i^*} means that the parity computed by X will differ on the two inputs. Since X rejects σ' and only depends on the z_i variables, X becomes false under the restriction $z_i = \text{true}\ \forall i \neq i^*$ and $z_{i^*} = \text{false}$.

Because P contains an XOR gate which becomes false under the restriction $z_i = \text{true}\ \forall i \neq i^*$ and $z_{i^*} = \text{false}$, P becomes false under this restriction as well. Since P was chosen arbitrarily and i^* does not depend on P, every pseudoproduct in U becomes false under this restriction. □

Let S^* be the restriction of S' to $z_i = \text{true}\ \forall i \neq i^*$ and $z_{i^*} = \text{false}$. By Claim 2.7.1, S^* only depends on pseudoproducts outside of U. Also, S^* must be equivalent to D, since we have restricted such that $z_1 \wedge \cdots \wedge z_\ell$ is false and $S' \equiv S = D \vee \bigvee_i (z_1 \wedge \cdots \wedge z_\ell \wedge \neg x_i)$. Thus, since $S^* \equiv D$, $|S^*|_w \geq \tau$.

Claim 2.7.2. *The total weighted size of the pseudoproducts in U is greater than $k(k+1)\ell$.*

Proof. First, note that the pseudoproducts in U depend on each z_i since any assignment accepted by S' but not by D must have all z_i true. This follows from the fact that S' is equivalent to $S = D \vee \bigvee_i (z_1 \wedge \cdots \wedge z_\ell \wedge \neg x_i)$. Thus, if we change a z_i from true to false in an assignment accepted by a pseudoproduct in U but not by D, the pseudoproduct must become false and therefore contains z_i. Thus, to show that the total weighted size is strictly greater than $k(k+1)\ell$, we need only show that the contributions to the weighted size from variables other than the z_i is greater than or equal to $k(k+1)\ell$.

If the pseudoproducts in U contain even a single v_i variable, the size contribution is at least $2k^2\ell \geq k(k+1)\ell$ for all $k \geq 1$. So the claim is proven in this case.

Otherwise, the pseudoproducts in U contain only x_i and z_i variables. Now we will find the size contribution of the x_i variables contained in U. Let U' be the set of all members of U, restricted to $z_i = \text{true}\ \forall i$. The disjunction of the pseudoproducts in U' must accept everything not accepted by D other than the all true assignment, since by the definition of U none of the pseudoproducts outside of U accept anything not accepted by D, and S' accepts everything but the all true assignment under the restriction $z_i = \text{true } \forall i$.

By Lemma 2.5, a minimum formula accepting everything not accepted by D and rejecting the all true assignment to the x_i variables is of the form $\bigvee_{i \in I} \neg x_i$. Since the MSSC instance is negative, i.e. $|I| > k$, there are at least $k+1$ x_i variables of weight $k\ell$ each, and thus the total size of the pseudoproducts in U' is at least $(k+1)k\ell$. Thus, the size contribution of x_i variables in U is at least $k(k+1)\ell$, which completes the proof of the claim. □

The total size of S' can now be calculated. The total size of all the pseudoproducts outside of U is at least τ. By Claim 2.7.2, the total size of the pseudoproducts in U is greater than $k(k+1)\ell$. Thus,

$$|S'|_w > \tau + k(k+1)\ell,$$

contradicting $|S'|_w \leq \tau + k(k+1)\ell$. So a negative instance of MSSC implies a negative instance of W-MSF, completing the proof. □

Thus, W-MSF is Σ_2^P-complete under Turing reductions, and therefore so is MSF by Theorem 2.4.

3 Conclusions and open problems

In this paper, we have resolved several problems relating to the complexity of SPP formulae. We have proven that formula equivalence remains coNP-complete and that irredundancy remains Σ_2^P-hard to approximate to within n^ϵ for some $\epsilon > 0$ when we generalize DNF formulae to SPP formulae. We also saw that when we generalize IS-PRIMI to SPP-PP, the DP-hardness result remains. Since we do not show SPP-PP $\in$ DP, the complete characterization of SPP-PP remains an open problem. Most importantly, we have resolved the complexity of the MSF problem, providing critical theoretical background to an important area of research in the field of logic synthesis.

The reduction used in this paper is a Turing reduction. A Turing reduction is needed to find the minimum formula size for D, which is critical to the reduction. So finding a many-one reduction remains an open problem.

Approximability of SPP minimization has not been addressed here, and remains an interesting open problem. The nature of the reduction given in this paper precludes direct use toward an inapproximability result. This is because the way in which the variables are weighted causes the part of the formula computing D to vastly outweigh the part determining how many x_i variables are necessary. This makes it impossible to use the reduction to approximate the number of x_i variables necessary. If this could be reversed such that each x_i variable carried greater weight in the problem size than the portion computing D, it would lead to both approximability results and a many-one reduction.

Rather than attempt to find a minimum SPP formula, several researchers have further restricted the possible solution space by restricting the fan-out of the XOR gates to a constant [BCDV08, CB02, Cir03].

Definition 3.1 (k-SPP Formulae)**.** *A k-SPP formula is an SPP formula in which the XOR gates have fan-out of at most k.*

While a k-SPP formula may require larger size than an SPP formula, they are sufficiently powerful to be exponentially smaller than the smallest equivalent DNF in some cases [Cir03].

Although k-SPP formulae are an active area of research [BCDV08], we have not resolved their complexity here. However, this work is an important first step toward determining the complexity of k-SPP, since so little was known about the complexity of either SPP or k-SPP minimization.

References

[BCDV08] Anna Bernasconi, Valentina Ciriani, Rolf Drechsler, and Tiziano Villa. Logic minimization and testability of 2-SPP networks. 2008. to appear in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[BU08] David Buchfuhrer and Christopher Umans. The complexity of Boolean formula minimization. In *ICALP*, pages 24–35, 2008.

[CB02] Valentina Ciriani and Anna Bernasconi. 2-SPP: a practical trade-off between SP and SPP synthesis. In *5th International Workshop on Boolean Problems (IWSBP2002)*, pages 133–140, 2002.

[Cir03] Valentina Ciriani. Synthesis of SPP three-level logic networks using affine spaces. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(10):1310–1323, 2003.

[GHM08] Judy Goldsmith, Matthias Hagen, and Martin Mundhenk. Complexity of DNF minimization and isomorphism testing for monotone formulas. *Information and Computation*, 206(6):760–775, June 2008. to appear.

[JSA97] James Jacob, P. Srinivas Sivakumar, and Vishwani D. Agrawal. Adder and comparator synthesis with exclusive-or transform of inputs. In *VLSI Design*, pages 514–515. IEEE Computer Society, 1997.

[LP99] Fabrizio Luccio and Linda Pagli. On a new Boolean function with applications. *IEEE Trans. Computers*, 48(3):296–310, 1999.

[Uma98] C. Umans. The minimum equivalent DNF problem and shortest implicants. In *FOCS*, pages 556–563, 1998.

[Uma99] Christopher Umans. Hardness of approximating Σ_2^p minimization problems. In *FOCS*, pages 465–474, 1999.

[Uma01] Christopher Umans. The minimum equivalent DNF problem and shortest implicants. *J. Comput. Syst. Sci.*, 63(4):597–611, 2001.

[UVSV06] C. Umans, T. Villa, and A. L. Sangiovanni-Vincentelli. Complexity of two-level logic minimization. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(7):1230–1246, 2006.