The complexity of formula minimization

Thesis by

David Buchfuhrer

In Partial Fulfillment of the Requirements
for the Degree of
Master of Science

California Institute of Technology
Pasadena, California

2008
(Submitted May 29, 2008)

Acknowledgements

Thanks to Chris Umans for help obtaining and writing up the results contained herein.

An extended abstract version of the material presented in Chapter 2 is set to appear in ICALP in 2008 [BU08].

Abstract

The Minimum Equivalent Expression problem is a natural optimization problem in the second level of the Polynomial-Time Hierarchy. It has long been conjectured to be Σ_2^P-complete and indeed appears as an open problem in Garey and Johnson [GJ79]. The depth-2 variant was only shown to be Σ_2^P-complete in 1998 [Uma98], and even resolving the complexity of the depth-3 version has been mentioned as a challenging open problem. We prove that the depth-k version is Σ_2^P-complete under Turing reductions for all $k \geq 3$. We also settle the complexity of the original, unbounded depth Minimum Equivalent Expression problem, by showing that it too is Σ_2^P-complete under Turing reductions.

We also consider 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 has been known about the complexity of SPP minimization. Here, we show that SPP minimization is Σ_2^P-complete under Turing reductions.

Contents

Chapter 1

Introduction

Circuit minimization problems are natural optimization problems contained in the second level of the Polynomial-Time Hierarchy (PH). The general form of such a problem is: given a Boolean circuit, find the smallest Boolean circuit that computes the same function. The input and output circuit may be required to be circuits of a particular form, e.g., Boolean formulas, or bounded-depth circuits. These problems are central problems in the field of logic synthesis, where fairly large instances are routinely solved using heuristics [DGK94]. They are also the prime examples of natural problems that should be complete for the classes of the second level of the PH. Indeed, versions of these problems inspired the definition of the PH in the early 70s by Meyer and Stockmeyer [MS72, Sto76], and Garey and Johnson use the formula variant to motivate the definition of the second level of the PH [GJ79].

Completeness proofs for circuit minimization problems have been hard to find. The DNF version of circuit minimization was only proven to be Σ_2^P-complete in 1998 by Umans [Uma98]; the other variants have remained prominent open problems. The only non-trivial hardness result for the formula variant – called Minimum Equivalent Expression – is a $P_{||}^{NP}$-hardness result of Hemaspaandra and Wechsung in 1997 [HW97]. One reason reductions for these problems are difficult is that one direction of the reduction entails proving a lower bound for the type of circuit under consideration. This shouldn't be a absolute barrier, though, for two reasons. First, we have lower-bound proof techniques for Boolean formulas and bounded-depth circuits; nevertheless incorporating these into a reduction seems tricky. Second, a reduction need not entail *strong* lower bounds and in principle even slightly non-trivial lower bounds could suffice. A similar difficulty for potential reductions showing the (conjectured) NP-hardness of a related problem was noted by Cai and Kabanets [KC00], although there, the use of weak lower bounds is not even an option, under a complexity assumption.

Proving Σ_2^P-completeness of the depth-3 variant was proposed [UVSV06] as a challenging first step, one that might begin to utilize techniques for proving lower bounds for bounded depth circuits (e.g., the Switching Lemma). In Chapter 2 we resolve, in one shot, the depth-3 case, as well as the depth-k variants for all $k \geq 3$. The same techniques show in addition that the unbounded depth

Minimum Equivalent Expression problem is Σ_2^P-complete under Turing reductions. We are able to achieve our results by exploiting the second way around the apparent barrier of proving circuit lower bounds: our reductions entail circuit lower bounds, but we get by with very weak ones, that with some effort are incorporated naturally into the structure of the reduction.

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 1.0.1 (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 on 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. Essentially, the number of true inputs 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.

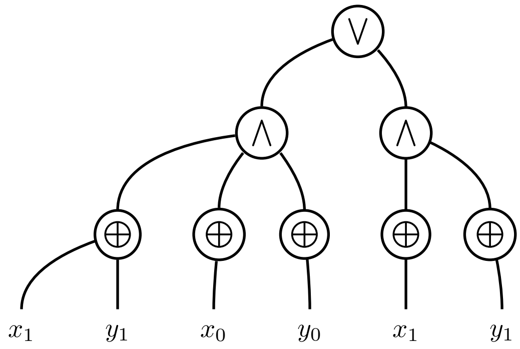

Figure 1.1: An SPP formula for the high bit output of a 2-bit adder.

Definition 1.0.2 (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 Chapter 3.

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. 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 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 1.0.1 (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?*

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 1.0.3 (k-SPP Formulae)**.** *An SPP formula in which the XOR gates have fan-out of at most k is referred to as a k-SPP formula.*

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 do not resolve their complexity here. However, this work is an important first step toward determining the complexity of k-SPP, since so little is known theoretically about either SPP or k-SPP minimization.

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 the knowledge of circuit minimization. Σ_2^P hardness 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. Σ_2^P hardness demonstrates that polynomial time algorithms making use of NP or coNP queries cannot exist unless the PH collapses. Σ_2^P hardness of MSF is especially important since SPPs are well-suited for use in practical situations such as representation of arithmetic functions [LP99, JSA97].

Chapter 2

Minimum Equivalent Expression

In this chapter, we prove that the Minimum Equivalent Expression problem is Σ_2^P-complete for both constant and unlimited depth formulae.

2.1 Description of the reduction

In this section we give a high-level description of the reduction, emphasizing a few interesting features before delving into the technical details.

The problem we reduce from is SUCCINCT SET COVER, which was defined and shown to be Σ_2^P-complete in [Uma99b]:

Problem 2.1.1 (SUCCINCT SET COVER (SSC))**.** *Given a DNF formula D on variables*

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

and an integer k, is there a subset $I \subseteq \{1, 2, \ldots n\}$ with $|I| \leq k$ and $D \vee \bigvee_{i \in I} \overline{x_i} \equiv 1$?

This can be seen as a succinct version of Set Cover, in which the sets are implicitly specified by the ones of the formulas D, $\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}$, and D is mandatory in any set cover (e.g., because it covers some point not covered by any of the other sets).

We assume that the formula D accepts the all-true assignment, as it only requires polynomial time to check this and the SSC instance is trivially false otherwise.

Our reductions exploit the special structure of this "succinct" set cover instance. In particular, all of the sets other than the one implicitly specified by D have an extremely simple form (they are just halfspaces), and in our reduction to MINIMUM EQUIVALENT EXPRESSION the choice of whether they are included or excluded from a cover will manifest itself relatively easily in the size of minimum equivalent expression. However, D may be a complicated function, one whose minimum formula size is not readily apparent. To circumvent this problem, we will use a Turing Reduction (actually a non-adaptive, or *truth-table*, reduction) which first ascertains the minimum formula size of D, and then asks one further query on a formula that incorporates D and other components, to determine whether or not the original instance of SUCCINCT SET COVER is a positive instance. This provides a somewhat rare example of a natural problem for which a Turing reduction seems crucial (in the sense that we do not know of any simple modification or alternative methods that would give a many-one reduction).

More specifically, the main idea of our reduction is to consider the following formula, derived from an instance of SUCCINCT SET COVER:

$$D \vee [z \wedge (\overline{x_1} \vee \cdots \vee \overline{x_n})] \tag{2.1}$$

where z is a new variable. Notice that when z is *false*, this formula is equivalent to just D, which (intuitively) forces a minimum equivalent formula to devote part of its size to computing an "unpolluted" copy of D. When z is *true*, the formula covers exactly the union of all of the "sets" in the instance of SUCCINCT SET COVER. That problem asks whether the disjunction of k or fewer $\overline{x_i}$ literals suffice to accept everything not accepted by D. If the variables indexed by $I \subseteq \{1, 2, \ldots, n\}$ suffice, then a very economical equivalent formula to the one above is

$$D \vee \left[z \wedge \left(\bigvee_{i \in I} \overline{x_i} \right) \right].$$

By forcing a minimum equivalent formula to contain a copy of D, we can ensure that the most efficient way to compute the above equivalent formula is indeed of this intended form. We can then determine whether or not there is a cover of size k by asking whether (2.1) has an equivalent formula of size at most k greater than the size of the D sub-formula and the z variable.

To make this actually work requires some modifications. For example, because the sets in the original instance cover all points in the domain, $D \vee z$ is already a small equivalent formula which does not depend at all on whether or not the instance of SUCCINCT SET COVER was a positive instance. But, we can solve this problem by modifying D initially to not accept the assignment in which every variable is set to *true*.

A more general technique that we use in several places in the reductions is "weighting" some variables in order to control the form of candidate small equivalent expressions. This is accomplished by replacing a single variable y with a conjunction of new variables, $y_1 \wedge \cdots \wedge y_w$, where w is the desired weight. We show that after this replacement, a minimum equivalent expression must be at least as large as the "w-minimum" expression (in which the size of a formula is measured by the number of occurrences of variables other than y plus w times the number of occurrences of the variable y).

We use this technique, for example, to weight z so highly that there can be only one occurrence of it; this then forms the conceptual pivot from which we argue that the subtrees of the formula surrounding that occurrence of z must compute D, and separately, a disjunction of as many variables as there are sets in a minimum cover of the original SUCCINCT SET COVER instance.

2.2 Outline.

In Section 2.3 we define general notation, and the variants of the problems we will be considering. In Section 2.5 we give the reductions – first a reduction showing that we can demand that the top-gate be an OR gate (or an AND gate) in Subsection 2.5.1, and then the main reductions in Subsection 2.5.2. We conclude in Section 2.6 with some open problems.

2.3 Preliminaries

Given a Boolean formula F, we use $|F|$ to mean the size of the formula F, and $\overline{F}$ for the negation of the formula F. Similarly, for a variable x, $\overline{x}$ is the negation of x.

Restrictions. Given a function $f : \{\mathit{true}, \mathit{false}\}^n \rightarrow \{\mathit{true}, \mathit{false}\}$ and a function $\rho : [n] \rightarrow \{\mathit{true}, \mathit{false}, \mathit{free}\}$, we define the *restriction* of f to ρ, f_ρ to be the function which fixes the ith input to $\rho(i)$ if $\rho(i)$ is not equal to *free*, and leaves it as an input otherwise. Similarly, if F is a formula for f, we define F_ρ to be the formula in which every instance of the ith input variable is replaced with $\rho(i)$ if $\rho(i) \neq \mathit{free}$, and is unchanged otherwise. Note that F_ρ is a formula for f_ρ.

Weighted formulae. If the variables x_i of some function f have associated weights $w(x_i)$, then the *w-weighted size* of a formula for f is the sum of the weights of the variables occurring at the leaves (in their multiplicity). The usual measure of formula size is the w-weighted size when $w(x_i) = 1$ for all x_i. Note that, as usual, size counts the number of literals at the leaves, and not the $(\vee, \wedge, \neg)$ gates.

Given a weight function w, we can take a formula F and create a formula F' which has minimum formula size that is at least the minimum w-weighted formula size of F. Formula F' is obtained by substituting $x_i^{(1)} \wedge x_i^{(2)} \wedge \cdots \wedge x_i^{(w(x_i))}$ for every occurrence of x_i in F. Note that by moving negations to the variable level, we are substituting $\overline{x_i^{(1)}} \vee \overline{x_i^{(2)}} \vee \cdots \vee \overline{x_i^{(w(x))}}$ for every occurrence of $\overline{x_i}$. We call F' the *w-expanded version* of F. The following lemma demonstrates the usefulness of this transformation:

Lemma 2.3.1. *Let F be a formula and w a weight function for F. Let F' be the w-expanded version of F. Then the minimum size of a formula equivalent to F' is at least the minimum w-weighted size of a formula equivalent to F.*

Proof. Consider a minimum formula $\widehat{F}'$ equivalent to F'. For each x_i, let $1 \leq j_i \leq w(x_i)$ be the integer for which $x_i^{(j_i)}$ occurs least among the x_i-leaves of $\widehat{F}'$. Consider the restriction ρ that for each i sets $x_i^{(j)}$ to *true* for $j \neq j_i$. By our choice of j_i, $|\widehat{F}'|$ is at least the w-weighted size of $\widehat{F}'_\rho$. But the formula $\widehat{F}'_\rho$ clearly is equivalent to F, so its w-weighted size is an upper bound on the minimum w-weighted size of a formula equivalent to F. □

2.4 The problems

As mentioned in the introduction, we will reduce from the Σ_2^P-complete problem SUCCINCT SET COVER. It will be convenient to work with a slightly modified version in which the goal is for the succinctly specified sets to cover everything *except* the "all-true" assignment.

Problem 2.4.1 (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

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

It's easy to see that this variant of SUCCINCT SET COVER is Σ_2^P-complete by reducing from SUCCINCT SET COVER:

Theorem 2.4.1. MSSC *is Σ_2^P-complete under Turing reductions.*

Proof. We are given an instance of SSC: a DNF D on variables

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

and an integer k. We produce the instance

$$D' = D \wedge \left(\bigvee_{i=1}^{m} \overline{v_i} \vee \bigvee_{i=1}^{n} \overline{x_i} \right)$$

(multiplied out into DNF) paired with the same integer k. If there exists $I \subseteq [n]$ of size at most k for which $D \vee \bigvee_{i \in I} \overline{x_i} \equiv 1$ then clearly $D' \vee \bigvee_{i \in I} \overline{x_i}$ accepts everything except the "all-true" assignment, and vice-versa. □

Remark 1. *In both* SSC *and* MSSC*, the instances produced by the reduction have the property that taking $I = \{1, 2, \ldots, n\}$ is a feasible solution.*

The central problem we are concerned with in Chapter 2 is:

Problem 2.4.2 (MINIMUM EQUIVALENT EXPRESSION (MEE))**.** *Given a Boolean $(\wedge, \vee, \neg)$-formula F and an integer k, is there an equivalent $(\wedge, \vee, \neg)$-formula of size at most k?*

We also consider the constant-depth versions. When discussing constant-depth formulas, as usual, we allow arbitrary fan-in AND and OR gates and we use the convention that all NOT gates occur at the variable level.

Problem 2.4.3 (MINIMUM EQUIVALENT DEPTH d EXPRESSION (MEE_d))**.** *Given a depth d Boolean formula F and an integer k, is there an equivalent depth d formula of size at most k?*

While distributing the NOT gates to the variable level clearly does not affect formula size, it's not as clear that finding the minimum depth-d formula is equivalent to finding the minimum depth-d formula with an OR gate at the root. The latter variant, defined below, will be easier to work with.

Problem 2.4.4 (MINIMUM EQUIVALENT DEPTH d EXPRESSION WITH TOP OR GATE (MEE_d−OR))**.** *Given Boolean formula F and an integer k, is there an equivalent depth d formula with top OR gate, of size at most k?*

In Theorem 2.5.3 we reduce MEE_d−OR to MEE_d, so that Σ_2^p-hardness for the latter follows from the Σ_2^p-hardness for the former.

2.5 Main results

In this section we prove:

Theorem 2.5.1. *For every $d \geq 3$, the problem MEE_d is Σ_2^P-complete under polynomial time Turing reductions.*

Theorem 2.5.2. *The problem MEE is Σ_2^P-complete under polynomial time Turing reductions.*

These two theorems are proved via the reductions in Sections 2.5.1, 2.5.2, and 2.5.3. The first allows us to restrict our attention to the MEE_d−OR problem, rather than the general MEE_d problem.

2.5.1 Top OR gate vs. unrestricted top gate

Theorem 2.5.3. *For every $d \geqslant 3$, there is a polynomial-time reduction from MEE_d−OR to MEE_d.*

Proof. Fix $d \geqslant 3$. If every depth-d formula F has a minimum equivalent depth-d formula F' with an OR gate at the root, then the two problems are equivalent, and the identity reduction suffices.

Otherwise there exists a formula F^* such that F^* has a smaller equivalent depth-d formula with an AND at the root than the smallest equivalent depth-d formula with an OR at the root. Equivalently, $\overline{F^*}$ has a smaller equivalent depth-d formula with an OR at the root than the smallest equivalent depth-d formula with an AND at the root. Let G be a minimum formula for $\overline{F^*}$.

Now, given a depth-d formula F and an integer k (which we may assume to be less than $|F|$), we create $|F|+1$ copies of G on disjoint variable sets (also disjoint from the variable set of F). Call these copies G_i. Our reduction produces the formula

$$F' = F \vee G_1 \vee \cdots \vee G_{|F|+1} \tag{2.2}$$

paired with the integer $k' = (|F|+1)|G| + k$.

If F has an equivalent depth-d formula with an OR at the root, of size at most k, then it is clear that F' has an equivalent depth-d formula of size at most k'.

If F does not have an equivalent depth-d formula with an OR gate at the root, of size at most $k \geq 0$, then we note that it cannot be a constant function. We wish to show that in this case F' does not have an equivalent depth-d formula of size at most k'. Suppose for the purpose of contradiction that it did, and call the equivalent formula $\widehat{F'}$. We claim that $\widehat{F'}$ must have an OR gate at the root. Note that G cannot be a constant function. Therefore, for each i there is a restriction ρ_i that sets the variables of F so that F evaluates to 0, and sets the variables of G_j for $j \neq i$ so that G_j evaluates to 0. The resulting formula $\widehat{F'}_{\rho_i}$ is equivalent to G_i, and if $\widehat{F'}$ had an AND gate at the root, this would be a depth-d formula for G_i with an AND gate at the root, which must have size at least $|G|+1$. This holds for each i, and the G_i are on disjoint variable sets, so the total size of $\widehat{F'}$ must be at least $(|F|+1)(|G|+1)$, which is greater than k' (since we assumed that $k \leq |F|$).

Thus, $\widehat{F'}$ has size at most k' and an OR gate at the root. Since the G_i and F are not constant functions, and they are all on disjoint sets of variables, we can apply the restriction argument above to conclude that $(|F|+1)|G|$ leaves must be used to account for the various G_i, and then at most $k' - (|F|+1)|G| = k$ are available to compute F. Thus there must be an equivalent formula for F with an OR gate at the root, of size at most k (and this formula can be obtained by restricting the variables belonging to the G_i in $\widehat{F'}$ so that each G_i becomes 0). So we conclude that F has an equivalent depth-d formula with an OR gate at the root of size at most k, a contradiction. □

2.5.2 Main reduction

The following is a Turing Reduction from MSSC to MEE$_d-$OR. We describe the steps of a Turing Machine with access to an oracle for MEE$_d-$OR:

- We are given an instance of MSSC: a DNF D on variables

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

  and an integer k. Let w be the weighting function with $w(x_i) = 1$ for all i and $w(v_i) = n + 1$ for all i, and let D' be the w-expanded version of D. Note that D' has only depth 3, as we are expanding a DNF formula.

- We make at most $\log |D'|$ calls to the oracle to find the size u of the smallest equivalent depth-d formula with top OR gate for D', using binary search.

- Define the formula E involving fresh variables y_i and z as follows:

$$E = D \vee [(\overline{x_1} \vee \overline{x_2} \vee \cdots \vee \overline{x_n} \vee \overline{y_1} \vee \cdots \vee \overline{y_{u+n}}) \wedge z].$$

  Let w' be the weighting function with $w'(x_i) = 1$ for all i, $w'(v_i) = n + 1$ for all i, $w'(y_i) = 1$ for all i and $w'(z) = 2u + k + n + 1$, and let F be the w'-expanded version of E. We will label the "copies" of z used in the expanded version $z_1, z_2, \ldots, z_{2u+k+n+1}$. Note that F has only depth 3.

- We ask the oracle if F has an equivalent depth-d formula with top OR gate, of size at most $4u + 2k + 2n + 1$. We will show that the answer is "yes" iff the original MSSC instance was a positive instance.

Remark 2. *Note that since this reduction utilizes logarithmically many adaptive oracle calls, it can be transformed using standard techniques into a non-adaptive reduction utilizing polynomially many oracle calls.*

The remainder of this section is devoted to proving the following theorem:

Theorem 2.5.4. *Let $\widehat{F}$ be a minimum equivalent depth-d formula with top OR gate for F. Then $|\widehat{F}| \leqslant 4u + 2k + 2n + 1$ iff there exists $I \subseteq \{1, 2, \ldots, n\}$ with $|I| \leq k$ and and for which*

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

As a point of reference, Figure 2.1 shows the "intended" form of a minimum equivalent depth-d formula for F. Of course for one direction of the reduction we will need to show that a small formula must have this form, which is a somewhat involved argument.

In the forward (easy) direction, we claim that if the instance of MSSC is a positive instance, then there is a depth-d formula equivalent to F, of the form pictured in Figure 2.1, and with size at

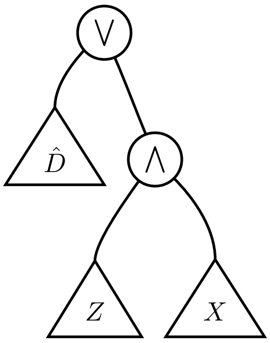

Figure 2.1: The desired form of an equivalent formula for F. Here $\widehat{D}$ is a minimum depth-d formula with top OR gate equivalent to D', $Z = \bigwedge_{i=1}^{2u+k+n+1} z_i$, and X is of the form $\bigvee_{i \in I} \overline{x_i} \vee \bigvee_{i=1}^{u+n} \overline{y_i}$.

most $4u + 2k + 2n + 1$. Let $\widehat{D}$ be a depth-d formula with top OR gate equivalent to D' of size u, and let $I \subseteq \{1, 2, \ldots, n\}$ be a set of size at most k for which

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

(such a set I exists because the MSSC instance is a positive instance). Then

$$\widehat{D} \vee \left[\left(\bigvee_{i=1}^{2u+k+n+1} z_i \right) \wedge \left(\bigvee_{i \in I} \overline{x_i} \vee \bigvee_{i=1}^{u+n} \overline{y_i} \right) \right]$$

is a depth-d formula equivalent to F of size $4u + 2k + 2n + 1$. Furthermore, it is of the form pictured in Figure 2.1.

In the other direction, we assume that the MSSC instance is a negative instance, and we wish to show that there is *no* depth-d formula with top OR gate equivalent to F of size at most $4u+2k+2n+1$. Let $\widehat{F}$ be a minimum depth-d formula for F.

We will derive from $\widehat{F}$ a minimum depth-d formula for F that is of the form pictured in Figure 2.1. We prove this in the next three subsections. Note that if $\widehat{F}$ has size larger than $4u+2k+2n+1$, then we are done; therefore we will assume the contrary in what follows.

2.5.2.1 The z variable.

First, we show that there is some i for which z_i occurs exactly once in $\widehat{F}$ and that it does not occur negated.

Lemma 2.5.5. *If a formula for F has size at most $4u + 2k + 2n + 1$, then there is some i such that z_i occurs exactly once.*

Proof. If the formula is of size at most $4u+2k+2n+1$ and yet contains two or more copies of each z_i, then this is a contradiction as the number of occurrences of z_i variables alone is $2(2u + k + n + 1) =$

$4u + 2k + 2n + 2 > 4u + 2k + 2n + 1$.

Thus, some z_i must occur at most once. Now, since we are assuming that D came from a negative instance of MSSC, we know that D rejects some assignment to its variables other than the all-true assignment. On the other hand E accepts this assignment when the z variable is true. This implies that E depends on z and that F (the w'-expanded version of E) depends on each z_i. So some z_i occurs *exactly* once. □

Fix an i for which z_i occurs exactly once. Now, let ρ be the restriction that restricts all z_j for $j \neq i$ to *true*, and leaves all other variables free. From now on we will be working with $\widehat{F}_\rho$, which has only a single z variable.

Lemma 2.5.6. *Let f be the function corresponding to $\widehat{F}_\rho$. Further, let ρ_0 restrict z_i to false, leaving all other variables free and ρ_1 restrict z_i to true, leaving all other variables free. If z_i appears negated in $\widehat{F}_\rho$, then $f_{\rho_1} \Rightarrow f_{\rho_0}$.*

Proof. Consider any restriction σ that assigns all variables except z_i to 0/1 and leaves z_i free. $(\widehat{F}_\rho)_\sigma$ takes in the single input $\overline{z_i}$. Since there are no negations other than at the variable level, $(\widehat{F}_\rho)_\sigma$ is monotone in $\overline{z_i}$. Thus, $f_\sigma(\mathit{true}) \Rightarrow f_\sigma(\mathit{false})$. Since this is true for all σ, $f_{\rho_1} \Rightarrow f_{\rho_0}$. □

Now, if we substitute *false* for z_i in $\widehat{F}_\rho$, the function that this formula computes is equivalent to D', and if we substitute *true*, it accepts everything except the "all-true" assignment to the x, y, and w-weighted v variables. Defining f, ρ_0, ρ_1 as in Lemma 2.5.6 (and again using the fact that D comes from a negative instance of MSSC) we have that $f_{\rho_1} \not\Rightarrow f_{\rho_0}$. Lemma 2.5.6 then tells us that z_i occurs non-negated in the formula $\widehat{F}_\rho$.

2.5.2.2 Properties of $\widehat{F}_\rho$.

In the remainder of the proof, we will use ALLTRUE as shorthand for the all-true assignment to the variables of $\widehat{F}_\rho$ – namely, the x variables, y variables, the w'-expanded v variables, and z_i. Similarly $\overline{ALLTRUE}$ refers to the function that accepts every assignment to those variables except ALLTRUE. For future reference, we record a few useful properties of $\widehat{F}_\rho$:

Lemma 2.5.7. *The following properties regarding $\widehat{F}_\rho$ hold:*

1. $|\widehat{F}_\rho| \leq |\widehat{F}| - (2u + k + n)$
2. *$\widehat{F}_\rho$ is a minimum formula*
3. *When z_i is true, $\widehat{F}_\rho$ is equivalent to the formula $\overline{ALLTRUE}$ with z_i set to true.*
4. *When z_i is false, $\widehat{F}_\rho$ is equivalent to D'.*

Proof. Property (1) follows from the observation in the proof of Lemma 2.5.5 that F depends on each z_j, so every z_j must appear at least once in $\widehat{F}$.

Property (2) holds because $\widehat{F}$ is a minimum formula. If $\widehat{F}_\rho$ was not a minimum formula, then we could take a smaller formula equivalent to $\widehat{F}_\rho$ and replace z_i with $\bigwedge_{j=1}^{2u+k+n+1} z_j$ to obtain a smaller formula for $\widehat{F}$.

Property (3) follows because for formula E, when z is true, E accepts exactly those assignments with at least one variable false. This property is preserved when the v variables are w'-expanded. The resulting function is the same as the one obtained by w'-expanding z and then restricting via ρ, after replacing z with z_i.

Property (4) follows from the definition of F and ρ. □

2.5.2.3 The X sub-formula.

In this section we show (Lemma 2.5.9) that a minimum formula accepting *at least* all of the assignments to the v, x and y variables not accepted by D' and not accepting the "all-ones" assignment is of the form

$$\left(\bigvee_{i\in I} \overline{x_i} \vee \bigvee_{i=1}^{u+n} \overline{y_i}\right).$$

We will eventually use this to argue that z_i's sibling sub-formula in $\widehat{F}_\rho$ has the intended form, and in a technical part of Section 2.5.2.4.

The following general lemma will be useful.

Lemma 2.5.8. *Let $t_1, t_2, \ldots, t_n$ be a set of variables, and S a subset of $\{0,1\}^n$. A minimum formula accepting at least S and not the all-true assignment is of the form $\bigvee_{i\in I} \overline{t_i}$ for some $I \subseteq \{1, 2, \ldots, n\}$.*

Proof. Let T be a formula accepting at least S and rejecting the all-true assignment. Suppose that T depends on ℓ variables. Then $|T| \geqslant \ell$. Furthermore, in each assignment accepted by T, one of these variables is set to *false*, as T does not accept all-true. Therefore, if T depends on variables t_i for $i \in I$, the formula $T' = \bigvee_{i\in I} \overline{t_i}$ accepts at least everything that T accepts. Furthermore T' does not accept all-true and $|T'| = \ell \leqslant |T|$.

Thus, given a minimum formula T that accepts at least S but not all-true, we can find another minimum formula accepting at least S but not all-true, of the desired form. □

Applying the lemma in our setting yields:

Lemma 2.5.9. *Let S be the set of assignments to the y variables plus the variables of D' (the w-expanded v variables and the x variables), that are not accepted by D'. Then a minimum formula accepting at least S but not the all-true assignment to these variables is of the form $\bigvee_{i\in I} \overline{x_i} \vee \bigvee_{i=1}^{u+n} \overline{y_i}$ for some $I \subseteq \{1, 2, \ldots, n\}$.*

Proof. By Lemma 2.5.8, we know that a minimum formula for this will be a disjunction of negated variables (from among the x, y and w-expanded v variables). Note that for each i, S includes the assignment in which y_i is *false*, all the other y variables are true, and all the other variables are true, because D' does not depend on y_i and it does not accept the all-true assignment to its variables. Therefore, the disjunction must accept this assignment. On the other hand, flipping y_i to *true* in this assignment results in the all-true assignment that the disjunction must not accept. Therefore the disjunction depends on each y_i, so it must contain all of the y variables.

Now, we simply need to see that none of the w-expanded v variables appear. If some $v_i^{(j)}$ does appear in the disjunction of negated variables, then it must be that D' rejects some assignment in which $v_i^{(j)}$ is *false* and every other variable in the disjunction is true (otherwise $v_i^{(j)}$ could be safely omitted). But then by symmetry, D' also rejects some assignment in which $v_i^{(j')}$ is *false* and every variable in the disjunction is true, for every $j' \neq j$ such that $v_i^{(j')}$ is not in the disjunction. Thus for all j', $v_i^{(j')}$ must be in the disjunction if $v_i^{(j)}$ is. So if a single v variable appears in the disjunction, then at least $n+1$ v variables appear (recall that $w(v_i) = n+1$ for all i). However, by Remark 1, we know that the disjunction of the n negated x variables together with all of the negated y variables suffices. This is a smaller disjunction than any disjunction involving v variables, which would need to include $n+1$ v variables (as argued above) together with all the y variables.

We conclude that a minimum formula accepting at least S but not the all-true assignment is of the claimed form. □

2.5.2.4 Position of the z variable.

Finally, we show (Lemma 2.5.13) that there is a minimum depth-d formula with top OR gate, equivalent to $\widehat{F}_\rho$, in which z_i occurs directly under a second-level AND gate. We begin with two general lemmas

Lemma 2.5.10. *Let A be a sub-formula of formula G, and suppose formula B implies G. Then, the formula obtained by replacing A with $A \vee B$ in G is equivalent to G.*

Proof. Because all of the negations have been pushed to the variable level, flipping the result of a non-input gate from *false* to *true* can only change the output of the formula from *false* to *true*, and not the reverse. Since we are replacing A with $A \vee B$, this can only change the result of the top gate of A from *false* to *true*. Furthermore, since B implies G, this can only occur when G is already true, and thus it will not change the output. □

Lemma 2.5.11. *Let A be a sub-formula of formula G that implies G. Then, the formula G' obtained by replacing A with false in G, and then taking the disjunction of this new formula with A is equivalent to G.*

Proof. In the case that A is true, then G must be true because A implies G. In this case G' will also be true, as A occurs directly beneath the top-level OR in G'. In the case that A is false, G' is equivalent to G with A replaced by *false*, so G' has the same result as G in this case as well. □

Note that the transformation in Lemma 2.5.11 does not increase the size of the formula, nor its depth if G already has a top OR gate. We now describe how the above general lemmas will be applied to $\widehat{F}_\rho$:

Lemma 2.5.12. *Suppose that $\widehat{F}_\rho$ has a sub-formula $A \wedge I$, as pictured in Figure 2.2. If $I \wedge \overline{ALLTRUE}$ implies $\widehat{F}_\rho$, then there is an equivalent depth-d formula with top OR gate, $\widehat{F}'_\rho$, where either $|\widehat{F}'_\rho| < |\widehat{F}_\rho|$ or $|\widehat{F}'_\rho| = |\widehat{F}_\rho|$ and $A \wedge I$ occurs at the second level.*

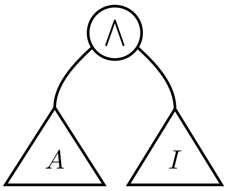

Figure 2.2: The sub-formula under consideration in Lemma 2.5.12

Proof. By assumption $I \wedge \overline{ALLTRUE}$ implies $\widehat{F}_\rho$, and so $A \wedge I \wedge \overline{ALLTRUE}$ does as well. If $A \wedge I \wedge \overline{ALLTRUE}$ is equivalent to $A \wedge I$, then $A \wedge I$ implies $\widehat{F}_\rho$. Then by Lemma 2.5.11, there is an equivalent formula $\widehat{F}'_\rho$ with $|\widehat{F}'_\rho| = |\widehat{F}_\rho|$ and $A \wedge I$ occurring at the second level.

Otherwise $A \wedge I$ accepts ALLTRUE. Since $A \wedge I$ accepts ALLTRUE, A must accept ALLTRUE, so $A \vee \overline{ALLTRUE} \equiv 1$. Thus, $I \wedge (A \vee \overline{ALLTRUE})$ is equivalent to I. Thus, by Lemma 2.5.10, we can replace $A \wedge I$ with I, resulting in a formula $\widehat{F}'_\rho$ which is equivalent to $\widehat{F}_\rho$ and $|\widehat{F}'_\rho| < |\widehat{F}_\rho|$. □

Lemma 2.5.13. *There is a depth-d formula with top OR gate, $\widehat{F}'_\rho$, that is equivalent to $\widehat{F}_\rho$ and of no larger size, in which z_i occurs exactly once, non-negated, and directly under a second-level AND gate.*

Proof. The proof is by case analysis. There are four cases depending on where z_i occurs in $\widehat{F}_\rho$: under the top-level OR gate, under an AND gate below the second level, under an OR gate below the third level, or under an OR gate at the third level.

Case 1: The variable z_i cannot occur directly under the top OR gate, as setting z_i to *true* would result in acceptance, and thus the formula would accept ALLTRUE, violating Lemma 2.5.7 (3).

Case 2: Suppose that z_i occurs under an AND gate below the second level. Then consider the sub-formula containing z_i, as pictured in Figure 2.3.

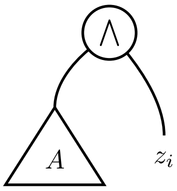

Figure 2.3: The portion of $\widehat{F}_\rho$ containing z_i if z_i is under a low-level AND gate

Since $z_i \wedge \overline{ALLTRUE}$ implies $\widehat{F}_\rho$, by Lemma 2.5.12 (with I set to z), we know that either $\widehat{F}_\rho$ is not minimal, contradicting Lemma 2.5.7 (2), or there is an equivalent depth-d formula with top OR gate, $\widehat{F}'_\rho$, with $|\widehat{F}'_\rho| = |\widehat{F}_\rho|$ and in which z_i occurs directly underneath a second-level AND gate.

Case 3: Suppose that z_i occurs under an OR gate below the third level. Then consider the sub-formula containing z_i, as pictured in Figure 2.4.

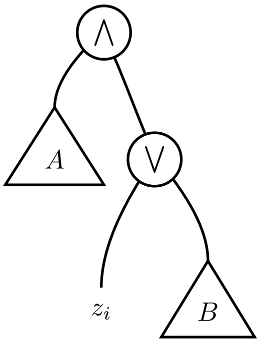

Figure 2.4: The portion of $\widehat{F}_\rho$ containing z_i if z_i is under a low-level OR gate

We will show that $(z_i \vee B) \wedge \overline{ALLTRUE}$ implies $\widehat{F}_\rho$. We already know that $z_i \wedge \overline{ALLTRUE}$ implies $\widehat{F}_\rho$, so we only need to see that $B \wedge \overline{ALLTRUE}$ implies $\widehat{F}_\rho$. Now, $(z_i \wedge B) \wedge \overline{ALLTRUE}$ implies $\widehat{F}_\rho$ because $z_i \wedge \overline{ALLTRUE}$ does. And, since z_i only occurs once in $\widehat{F}_\rho$, in disjunction with B, $(\overline{z_i} \wedge B) \wedge \overline{ALLTRUE}$ must also imply $\widehat{F}_\rho$, as flipping z_i from *true* to *false* will not change the result of any gate in the formula if B is already true. This is because setting B to *true* will satisfy the OR gate that z_i occurs under, and since z_i only occurs once it cannot affect the result of any other gate. Thus, $B \wedge \overline{ALLTRUE}$ implies $\widehat{F}_\rho$, as both $(z_i \wedge B) \wedge \overline{ALLTRUE}$ and $(\overline{z_i} \wedge B) \wedge \overline{ALLTRUE}$ do.

Thus, since $(z_i \vee B) \wedge \overline{ALLTRUE}$ implies $\widehat{F}_\rho$, by Lemma 2.5.12, we know that either $\widehat{F}_\rho$ is not minimal, contradicting Lemma 2.5.7 (2), or there is an equivalent depth-d formula with top OR gate, with size $|\widehat{F}_\rho|$ and in which this sub-formula occurs directly beneath a second-level AND gate, placing z_i directly underneath a third-level OR gate. Now we proceed through Case 4 below.

Case 4: If z_i occurs directly under a third-level OR gate, then the formula has the form in Figure 2.5

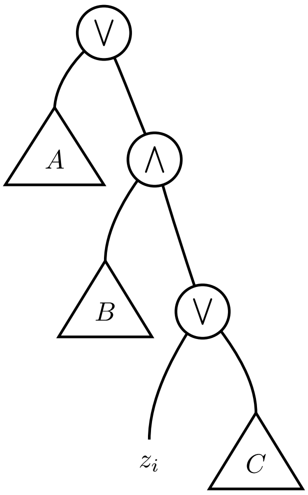

Figure 2.5: The form of F_i when z_i is under a 3^{rd}-level OR gate

Case 4a: Suppose that C does not accept ALLTRUE. Then we claim that C implies $\widehat{F}_\rho$. Consider an assignment accepted by $z_i \wedge C$. Since C does not accept ALLTRUE, such an assignment must have some variable false, and hence $\widehat{F}_\rho$ accepts it (by Lemma 2.5.7 (3)). However, flipping z_i to *false* in this assignment cannot alter the output of C (since it does not contain z_i) and therefore the OR gate above C remains true, and no other gate values above it change, since z_i occurs only once in the formula. Thus $\widehat{F}_\rho$ accepts this assignment as well. We conclude that $z_i \wedge C$ as well as $\overline{z_i} \wedge C$ imply $\widehat{F}_\rho$, and therefore C implies $\widehat{F}_\rho$.

Now, by Lemma 2.5.11, we can replace C with *false* and move C to the top-level OR gate. This leaves z_i alone under its OR gate, and so we can move it up one level so that it resides under the second level AND gate.

Case 4b: Suppose that C does accept ALLTRUE. Then we claim that C cannot accept anything else not accepted by D'. Suppose for the purpose of contradiction that it did, and let τ be such an assignment to the variables of D'. By Lemma 2.5.7 (4), when $z_i = \textit{false}$, $\widehat{F}_\rho$ does not accept τ, and by Lemma 2.5.7 (3), when $z_i = \textit{true}$, $\widehat{F}_\rho$ does accept τ (since τ is not the "all-true" assignment). However, toggling z_i in this assignment cannot alter the output of $\widehat{F}_\rho$, because z_i occurs only once, and under assignment τ, the sub-formula C already makes the OR above z_i true. This is a contradiction. Thus we know that $\overline{C}$ accepts at least everything not accepted by D', but not ALLTRUE, and then by Lemma 2.5.9, $\overline{C}$ has size at least $u + n$.

Referring again to Figure 2.5, we see that when restricting z_i to *true* in $\widehat{F}_\rho$, the formula reduces to $A \vee B$. By Lemma 2.5.7 (3), the resulting formula accepts everything except

ALLTRUE. Therefore $A \vee B$ depends on every variable other than z_i, and so every variable must appear at least once, and their combined size must be at least $u+n$ from the y variables alone.

Adding up the sizes of A, B, C and z_i, we have that $|\widehat{F}_\rho|$ is at least $(u+n)+(u+n)+1 = 2u+2n+1$. Applying Lemma 2.5.7 (1), we find that

$$|\widehat{F}| \geq (2u+2n+1)+(2u+k+n) = 4u+3n+k+1.$$

Since $k < n$ (the MSSC instance is trivially a positive instance if $k \geqslant n$) this quantity is strictly greater than $4u+2k+2n+1$, contradicting our original assumption that $|\hat{F}| \leqslant 4u+2k+2n+1$. We conclude that this sub-case cannot arise.

□

2.5.2.5 Finishing up.

Lemma 2.5.14. *There is a minimum depth-d formula with top OR gate equivalent to F, of the form pictured in Figure 2.6.*

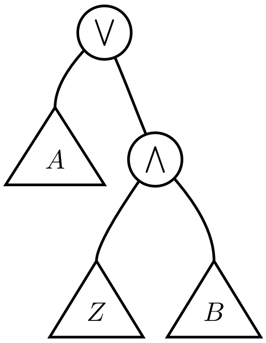

Figure 2.6: The required form of a minimum depth-d formula with top OR gate equivalent to F.

Proof. By Lemma 2.5.13, there is a minimum depth-d formula with top OR gate equivalent to $\widehat{F}_\rho$ that is of the form in Figure 2.6, but with the Z sub-formula replaced by z_i. Replacing z_i with $\bigwedge_{j=1}^{2u+k+n+1} z_j$, we obtain a depth-$d$ formula for F, of the form pictured in Figure 2.6, of size at most $|\widehat{F}_\rho| + 2u + k + n$. By Lemma 2.5.7 (1), this quantity is a lower bound on the size of a minimum depth-d formula with top OR gate equivalent to F, so it must be minimum. □

Now we are finally able to argue that $|\widehat{F}|$ must be larger than $4u+2k+2n+1$. First, observe that by Lemma 2.5.14, there is a depth-d formula equivalent to F of the form pictured in Figure 2.6 whose size is the same as the size of $\widehat{F}$. In this formula, when Z is set to false, the function simplifies to just A, and by the definition of F, this must be equivalent to D', and hence it must have size at least u. On the other hand, when Z is set to *true*, the formula must accept every assignment in which at least one variable is set to *false*. This means that B must accept everything not accepted by D' except the "all-true" assignment. By Lemma 2.5.9, we know that we can assume B to be a disjunction of negated variables, and that it must have size at least $u + n + k + 1$ (because the original MSSC instance was a negative instance). Adding the sizes of A and B to the number of z_i variables in Z, we have a formula of size at least $4u + 2k + 2n + 2$. We conclude that

$$|\widehat{F}| \geqslant 4u + 2k + 2n + 2 > 4u + 2k + 2n + 1$$

as required.

This completes the proof of Theorem 2.5.4.

2.5.3 The unbounded depth case

The reduction for MEE is the same as the reduction in the previous section (2.5.2). We never used the depth-d restriction in any of the arguments in that reduction; it was only mentioned in the context of ensuring that various manipulations *maintained* depth-d, a constraint that is no longer operative for the unbounded depth case.

It is still convenient in the reduction to think of formulas with alternating levels of unbounded-fan-in AND and OR gates, and here we simply note that discussing the size of such formulas is the same as discussing the size of standard, fan-in-2 $(\wedge, \vee, \neg)$-formulas.

Proposition 2.5.15. *If F is a formula with unbounded-fan-in AND and OR gates of size s, then there is an equivalent formula F' with fan-in-2 AND and OR gates of size s. Similarly if there is a formula F with fan-in-2 AND and OR gates of size s, there is an equivalent formula F' with unbounded-fan-in AND and OR gates of size s.*

2.6 Conclusions and open problems

The most natural open problems remaining are to give many-one reductions for the problems in this chapter (rather than Turing reductions), and to resolve the complexity of the circuit version of the problem. Our techniques here rely heavily on the fact that we are dealing with formulas rather than circuits.

Another important direction is to study the approximability of the problems in this chapter. For the depth-2 case (DNF minimization) it is known that the problems are inapproximable to within very large (N^{ε}) factors [Uma99a]. Our reductions are quite fragile and do not seem to give any hardness of approximation results for these problems.

Finally, we note that the complexity of the "Π_2^P" versions of all of these problems remain open. These are problems of the form: given a Boolean circuit (of some specified form), is it a minimum circuit (of the specified form). Even for depth two (DNFs), this problem is not known to be Π_2^P-complete, although it is conjectured to be complete for that class.

Chapter 3

Minimum SPP Formula

In this chapter, we explore results related to SPP formulae, including a proof that SPP formula minimization is Σ_2^P-complete.

3.1 Description of the reduction

This section contains a high-level description of the reduction used to show that Problem 1.0.1 is Σ_2^P-complete under Turing reductions, in order to facilitate understanding of a more technical description later. We also compare the reduction here to the one in Chapter 2.

As in Chapter 2, the problem we reduce from is Problem 2.4.1 Modified Succinct Set Cover (MSSC).

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 $\bigvee_i \overline{x_i}$. 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 $D \vee \bigvee_i \overline{x_i}$. With this variable added, the formula should look like $D \vee (z \wedge \bigvee_i \overline{x_i})$. Because SPP formulae only have one OR gate at the top level, the particular form is more similar to $D \vee \bigvee_i (z \wedge \overline{x_i})$.

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 $v_1 \oplus v_2 \oplus \cdots \oplus v_{w(v)}$, and the fact that this transformation behaves as if we had simply weighted a single variable v by $w(w)$ is shown by Lemma 2.3.1.

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 with variables $z_1, \ldots, z_\ell$.

The reduction approach we use in this chapter is superficially similar to one contained in Chapter 2, which proves that minimization of $(\vee, \wedge, \neg)$ formulae of both fixed and unrestricted depth are Σ_2^P-complete under Turing reductions. Consider the depth-3 version, which is most similar to SPP minimization.

Problem 3.1.1 (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 Chapter 2 showing that MEE$_3$ is Σ_2^P-complete under Turing reductions 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 $D \vee \bigvee_i \overline{x_i}$, playing a similar role to the z_i variables in this chapter, but with an altogether different form. In the SPP reduction shown in this chapter, we create a formula with a form similar to

$$D \vee \bigvee_i (z \wedge \overline{x_i}).$$

By contrast, in Chapter 2 the formula resulting from the reduction is more similar to

$$D \vee \left(z \wedge \bigvee_i \overline{x_i} \right).$$

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 Chapter 2, 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 $\bigvee_i \overline{x_i}$. Lemma 3.1.1 demonstrates why this prevents any such reduction from working.

Lemma 3.1.1. *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 3.1.1 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 same general approach as in Chapter 2. In a depth-3 Boolean formula like those under consideration in the earlier 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 the MEE reduction, 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 Chapter 2. 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 3.1.2.

Lemma 3.1.2. *If a pseudoproduct computes the conjunction of n variables $z_1, \dots, z_n$, the pseudoproduct contains at least n XORs.*

The proof of Lemma 3.1.2 is contained in Section 3.3.3 and utilizes simple linear algebra.

3.2 Outline

Section 3.3 contains all of our results. Section 3.3.1 contains some background results that carry over from complexity results for DNF formulae. Section 3.3.2 contains an explanation of weighting necessary to the main reduction. Section 3.3.3 contains the proof that Problem 1.0.1 is Σ_2^P-complete, as outlined in Section 3.1.

Finally, Section 3.4 contains a summary of the results as well as a discussion of related problems that remain open.

3.3 Results

In this section, we present new classifications for the complexity of problems related to SPP formulae. We begin in Section 3.3.1 with a few simple results, then continue in Section 3.3.3 with the proof of the Σ_2^P-completeness of MSF under Turing reductions.

3.3.1 Complexity of Related Problems

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

Problem 3.3.1 (SPP Equivalence)**.** *Given two SPP formula S, T, do both S and T compute the same function?*

Proposition 3.3.1. *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 3.3.2 (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 [Uma99a]. We see here that the corresponding problem for SPP formulae shares the same complexity classification.

Problem 3.3.3 (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 3.3.2. *SPP Irredundancy is Σ_2^P hard to approximate to within a factor of n^ε for some $\varepsilon > 0$, where n is the input size.*

Proof. As shown in [Uma99a], DNF Irredundancy is Σ_2^P-hard to approximate within n^ε for some $\varepsilon > 0$. As in the proof of Proposition 3.3.1, 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 3.3.1 (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 3.3.4 (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 3.3.2 (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 rather than minimum SPP formulae as defined in this paper [GHM08]. Still, they are important to the study of SPP formulae, so we consider the complexity of determining whether a pseudoproduct is prime.

Problem 3.3.5 (SPP Prime Pseudoproduct (SPP-PP))**.** *Given an SPP S and a pseudoproduct P, is P a prime pseudoproduct of S?*

We show that SPP-PP is DP-hard using the same reduction used in [GHM08] to prove that IS-PRIMI is DP-hard. The reduction is from SAT-UNSAT.

Problem 3.3.6 (SAT-UNSAT)**.** *Given CNF formulae α, β, is α satisfiable and β unsatisfiable?*

Theorem 3.3.3. *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. Lets now consider what 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 refer to the formulae taken as inputs, preserving 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 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.

One might also hope that the proof of DNF formula minimization being Σ_2^P-complete would directly carry over to a proof of SPP formula minimization being Σ_2^P-complete, as the proof for IS-PRIMI being DP-hard carried directly over to a proof of SPP-PP being DP-hard. 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. However, the proof of DNF minimization being Σ_2^P-complete does not have the similar feature that the DNF produced is a minimum DNF iff it is a minimum SPP. So a new reduction is necessary.

3.3.2 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 fan-out 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',

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

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 3.3.2. Recall that we represent $\neg x \oplus y$ by $x \oplus y \oplus 1$ in this paper.

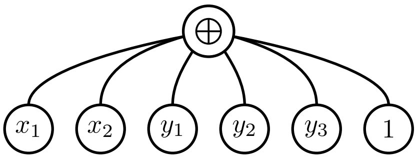

Figure 3.1: The expanded form of $\neg x \oplus y$ with weights $w(x) = 2, w(y) = 3$

The expansion described above differs from the expansion strategy used in the MEE reduction, 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 2.3.1 demonstrates that applying this simple transformation is equivalent to positive integer weighting.

Problem 3.3.7 (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 $|S'|_w \leqslant k$?*

We refer to a formula which is minimum with respect to w as a w-minimum formula.

Lemma 3.3.4. *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, $|F'| \leqslant |F|_w$, as the expanded version of F has size exactly $|F|_w$.

Now we will see that $|F'| \geqslant |F|_w$. 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 $|F'| \geqslant |F^*|_w$. Thus, since F is a w-minimum formula equivalent to F^*, we have $|F'| \geqslant |F^*|_w \geqslant |F|_w$.

Since $|F'| \leqslant |F|_w$ and $|F'| \geqslant |F|_w$, we have $|F'| = |F|_w$. □

Now, we apply Lemma 3.3.4 to show that W-MSF and MSF are poly-time equivalent.

Theorem 3.3.5. *There exist polynomial-time reductions between W-MSF and MSF.*

Proof. First, lets 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 $|S'|_w \leqslant k$, 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 3.3.4, 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.

3.3.3 Main Result

In this section, we show that Problem 1.0.1, 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 $\langle D, v_1, v_2, \ldots, v_m, x_1, x_2, \ldots, x_n, k\rangle$ of MSSC.
- Set $\ell = 2k(k+1)(|D|+1)$.
- Weight the v_i variables in D by $2k^2\ell$ and the x_i variables by $k\ell$.
- Using binary search, find the size of the w-minimum SPP formula for D, denoted τ, using $O(\log |D|_w)$ oracle calls.

- Set $S = D \vee (Z \wedge \neg x_1) \vee (Z \wedge \neg x_2) \vee \cdots \vee (Z \wedge \neg x_n)$ where $Z = z_1 \wedge z_2 \wedge \cdots \wedge z_\ell$ and the z_i are new variables of weight 1 and the x_i are weighted by $k\ell$ as mentioned above.

- Finally, we make one last oracle call to determine whether there is a weighted SPP formula for S of size at most $\tau + k(k+1)\ell$. Accept iff such an SPP exists.

Remark 3. *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 3.1.2.

Proof (of Lemma 3.1.2). 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 $w^{(i)} \in \mathbb{Z}_2^{n'}$ in which the jth position of $w^{(i)}$, $w^{(i)}_j$, is equal to 1 iff z_i is contained in the jth XOR.

Now we can match each assignment of truth values to the z_i variables to the sum $W(z) = \sum_i z_i w^{(i)}$. Note that the jth position of $W(z)$, $W(z)_j$, is equal to $\sum_i z_i w^{(i)}_j$, 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 constants it contains must be odd.

Thus, for P to evaluate to true, $W(z)_j$ should be equal to 1 iff the jth XOR contains an even number of true constants. 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 $W(z) = W^*$.

Since P computes a satisfiable function, there is some assignment to the variables $z_1, \ldots, z_n$ such that $W(z) = W^*$. However, since $w_1, \ldots, w_n$ 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 $w_1, \ldots, w_n$, 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. □

Although Lemma 2.5.8 was only proven for $(\vee, \wedge, \neg)$-formula, it holds true for weighted SPP as well using the same proof. Thus, we will be making use of Lemma 2.5.8 below.

One final necessary lemma will allow us to determine where the z_i variables are located.

Lemma 3.3.6. *Let S' be an equivalent formula to the SPP formula S created by the Turing reduction described at the beginning of Section 3.3.3. 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 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 $S = D \vee \bigvee_i (z_1 \wedge \cdots \wedge z_\ell \wedge \overline{x_i})$. 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 ${H_P}^C$. Thus, since P accepts σ and P' cannot accept any other assignment, P' computes exactly $\bigwedge_{z_i \in {H_P}^C} z_i$. Thus, by Lemma 3.1.2, 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 trivial 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, $|P|_w \geqslant \ell |{H_P}^C|$. 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

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

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

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

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

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

The main result that W-MSF is Σ_2^P-complete under Turing reductions, and therefore so is MSF, follows from Theorem 3.3.7.

Theorem 3.3.7. *Let S be the SPP formula resulting from the Turing reduction at the beginning of Section 3.3.3. 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 $|I| \leqslant k$ and $D \vee \bigvee_{i \in I} \overline{x_i} \equiv D \vee \bigvee_{i=1}^{n} \overline{x_i}$. 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' equivalent to S as follows.

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

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 Claim 3.3.7.1 to achieve a τ lower bound. We then show that the portion of the formula removed by the restriction is of size greater than $k(k+1)\ell$ by Claim 3.3.7.2.

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 $|S'|_w \leqslant \tau + k(k+1)\ell$. 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 3.3.6 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 3.3.6, there is some index i^* such that $z_{i^*} \in H_P \ \forall P \in U$, since $|S'|_w \leqslant \tau + k(k+1)\ell$.

Claim 3.3.7.1. *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 $S = D \vee \bigvee_i (z_1 \wedge \cdots \wedge z_\ell \wedge \overline{x_i})$. Consider the assignment σ' in which all variables are set according to σ, except for z_{i^*}, which is set to false, such that 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 3.3.7.1, 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 $S = D \vee \bigvee_i (z_1 \wedge \cdots \wedge z_\ell \wedge \overline{x_i})$. Thus, since $S^* \equiv D$, $|S^*|_w \geqslant \tau$.

Claim 3.3.7.2. *The total 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 S' only accepts things not accepted by D when all z_i are true. This follows from the fact that S' is equivalent to $S = D \vee \bigvee_i (z_1 \wedge \cdots \wedge z_\ell \wedge \overline{x_i})$. So this claim follows if we show that the variables other than the z_i contribute size at least $k(k+1)\ell$ to the total size of U.

If the pseudoproducts in U contain even a single v_i variable, the size contribution is at least $2k^2\ell \geqslant k(k+1)\ell$ for all $k \geqslant 1$. 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 under the restriction $z_i = \text{true}$.

By Lemma 2.5.8, a minimum formula accepting everything not accepted by D and rejecting the all true assignment to the x_i variables is of the form $\bigvee_{i \in I} \overline{x_i}$. Since the MSSC instance is negative, $|I| > k$, 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 3.3.7.2, the total size of the pseudoproducts in U is greater than $k(k+1)\ell$. Thus,

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

contradicting $|S'|_w \leqslant \tau + k(k+1)\ell$. 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.

3.4 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 $\varepsilon > 0$ 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.

Approximability of SPP minimization has not been addressed here, and remains an interesting open problem. The nature of the reduction given in this chapter 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 the removal of the need for a Turing reduction.

The k-SPP minimization problem is also left open here. The complexity of k-SPP minimization remains unresolved because the weighting scheme used in this chapter requires polynomial fan-out of the XOR gates. Without polynomial fan-out in the XOR gates, it is difficult to create a weighting scheme which does not cause exponential increase in the size of some SPP formulae.

Finally, we note that the full characterization of the complexity of SPP-PP remains unresolved, as we have proven that SPP-PP is DP-hard but not that SPP-PP $\in$ DP.

Bibliography

[BCDV08] A. Bernasconi, V. Ciriani, R. Drechsler, and T. 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] D. Buchfuhrer and C. Umans. The complexity of Boolean formula minimization. In *ICALP*, pages 24–35, 2008. to appear.

[CB02] V. Ciriani and A. 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] V. 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.

[DGK94] S. Devadas, A. Ghosh, and K. Keutzer. *Logic synthesis*. McGraw-Hill, Inc., New York, NY, USA, 1994.

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

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

[HW97] E. Hemaspaandra and G. Wechsung. The minimization problem for Boolean formulas. In *FOCS*, pages 575–584, 1997.

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

[KC00] V. Kabanets and J.-Y. Cai. Circuit minimization problem. In *STOC*, pages 73–79, 2000.

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

[MS72] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *FOCS*, pages 125–129. IEEE, 1972.

[Sto76] L. J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976.

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

[Uma99a] C. Umans. Hardness of approximating Σ_2^{p} minimization problems. In *FOCS*, pages 465–474, 1999.

[Uma99b] C. Umans. On the complexity and inapproximability of shortest implicant problems. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *ICALP*, volume 1644 of *Lecture Notes in Computer Science*, pages 687–696. Springer, 1999.

[Uma01] C. 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.