Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 81 (2024)

Circuits, Proofs and Propositional Model Counting

Sravanthi Chede[1], Leroy Chew[2], Anil Shukla[1]

[1]Indian Institute of Technology Ropar, Rupnagar, India
[2]TU Wien, Austria

August 2, 2024

Abstract

In this paper we present a new proof system framework CLIP (Circuit Linear Induction Proposition) for propositional model counting (#SAT). A CLIP proof firstly involves a Boolean circuit, calculating the cumulative function (or running count) of models counted up to a point, and secondly a propositional proof arguing for the correctness of the circuit.

This concept is remarkably simple and CLIP is modular so it allows us to use existing checking formats from propositional logic, especially strong proof systems. CLIP has polynomial-size proofs for XOR-pairs which are known to require exponential-size proofs in MICE [21]. The existence of a strong proof system that can tackle these hard problems was posed as an open problem in Beyersdorff et al. [5]. In addition, CLIP systems can p-simulate all other existing #SAT proofs systems (KCPS(#SAT) [12], CPOG [7], MICE). Furthermore, CLIP has a theoretical advantage over the other #SAT proof systems in the sense that CLIP only has lower bounds from its propositional proof system or if $\mathsf{P}^{\#\mathsf{P}}$ is not contained in P/poly, which is a major open problem in circuit complexity.

CLIP uses unrestricted circuits in its proof as compared to restricted structures used by the existing #SAT proof systems. In this way, CLIP avoids hardness or limitations due to circuit restrictions.

1 Introduction

Given a propositional formula, the problem of finding its total number of satisfying assignments (models) is known as the propositional model counting problem #SAT [30]. The problem is known to be #P-complete and is considered one of the hardest problem in the field of computational complexity. In fact, it is known that with an access to a #SAT-oracle, any problem from polynomial hierarchy can be solve in polynomial time (Toda's Theorem [33]).

Over the last few years, some important proof systems have been developed for #SAT. The KCPS(#SAT) [12] system is the first non-trivial proof system based on knowledge compilation designed for #SAT. A KCPS(#SAT) proof for a CNF represents the proposition as a decision-DNNF, with some additional annotations for checking. A decision-DNNF allows for model counting to be easily extracted. However, limitations and lower bounds for KCPS(#SAT) have already been established [3, 12]. The second proof system designed for #SAT is the MICE [21] proof system. Unlike KCPS(#SAT), it is a line based proof system which computes the model count in a step by step fashion using some simple inference rules. Several lower bounds for MICE have been established in the literature [3, 5]. For example, XOR-PAIRS [5], are shown to be hard for the MICE proof system. Recently, an important proof system CPOG [7] was introduced for #SAT. Similar to KCPS(#SAT), CPOG is also based on knowledge compilation. A CPOG proof for a CNF formula ϕ

ISSN 1433-8092

consists of a Partitioned-Operation Graph (POG) G along with a Resolution proof of the fact that $G \equiv \phi$.

The relationship between these three proof systems are now well known. It has been shown in [3], that CPOG is exponentially stronger than KCPS(#SAT) and MICE. On the other hand KCPS(#SAT) and MICE are incomparable [3, Figure 1]. This means that KCPS(#SAT) and MICE have unconditional lower bounds. For CPOG a lower bound is currently unknown, but the proof complexity is necessarily tied to the limitations of POGs.

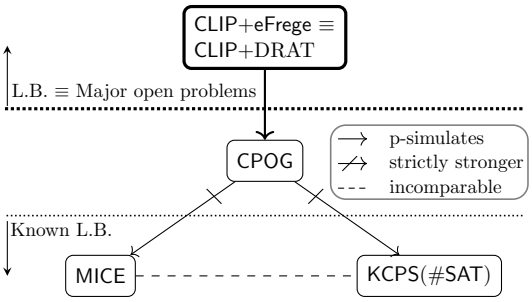


Figure 1: *Hierarchy of #SAT proof systems. New results are shown in bold.*

In this paper, we introduce a powerful #SAT proof format CLIP (Circuit Linear Induction Proposition) (Definition 5). A CLIP based proof for a CNF formula ϕ consists of a Boolean circuit calculating the running count of models up to an assignment treated in a lexicographical order. We denote this Boolean circuit as a cumulator (Definition 3). In addition to cumulator, the CLIP proof also contains a certificate proving the correctness of the cumulator. The CLIP format is similar to CPOG in the sense that instead of a POG, CLIP has a cumulator. Since POG uses restricted versions of AND and OR gates, as compared to cumulators, we believe that CLIP format is much stronger than CPOG. In this direction, we show that CLIP can p-simulate CPOG (Theorem 53). In fact, we show that CLIP has lower bounds only if some open problems of proof complexity or circuit complexity are solved (Theorem 7).

In MICE and KCPS(#SAT), proofs can grow exponentially because of unsatisfiable formulas that have lower bounds in Resolution. In this direction, for any unsatisfiable formula which is hard in Resolution, it is unclear how large the CPOG proofs will be. In our system, for unsatisfiable formulas, proofs are no bigger than the shortest DRAT proofs (Proposition 55). In addition, we also show that XOR-PAIRS which are known to be hard for existing proof systems are easy for the CLIP format (Theorem 65). We sum up our contributions in the Figure 1. We explain the same in detail in the following subsection.

1.1 Our Contributions

1. **Introducing a new proof system framework for #SAT (CLIP)**: We present a proof system where proofs are pairs containing a circuit and a propositional proof. The circuit is a multi-output Boolean circuit we call the *cumulator* which takes a complete assignment α and returns the number (in binary) of models of a propositional formula up to α in some fixed lexicographical ordering of assignments. In addition, CLIP proofs also contain a certificate showing the correctness of the cumulator. The certificate here is propositional proof. This is possible because we can construct a tautology that covers every inductive step for any two consecutive complete assignments. The CLIP format allows us to use any known propositional proof system $\mathcal{P}$ for proving the correctness of the cumulator. In this paper we focus on CLIP+ Extended Frege (CLIP+eFrege). We show that CLIP+eFrege is a powerful proof system in the sense that it has a lower bound only if a super-polynomial lower bound for eFrege is found or it is proven that $\mathsf{P}^{\#\mathsf{P}} \not\subseteq \mathsf{P/poly}$ (Theorem 7). Hence proving lower bounds in CLIP+eFrege will lead to solving major open problems in the fields of proof complexity or circuit complexity. Another way to say this is that, CLIP is the first #SAT proof system which is conditionally optimal. Note that such systems already exist in the propositional and QBF worlds, i.e. IPS

(the Ideal Proof System) [23] and eFrege +∀red [2] respectively.

2. **A CLIP+eFrege simulation technique for all existing #SAT proof systems**: The CLIP+eFrege simulation technique consists of three parts. The first part consists of extracting a cumulator from any #SAT proof system which is closed under restrictions. The second part establishes that if a #SAT proof system admits easy eFrege proofs of the properties of restriction (Definition 19) then it can be p-simulated by CLIP+eFrege. The final part proves that the CPOG system admits all the required properties and hence can be p-simulated by CLIP+eFrege. Let us briefly explain each of them separately.

   (a) **Cumulator extraction** (Section 4.1). We show that from any #SAT proof system which is closed under restrictions (Definition 1), there is a simple technique to efficiently extract a cumulator from its proofs. For this, we carefully use the concept of Fenwick trees [20] to introduce and compute the Fenwick assignments (Definition 13). Informally, the Fenwick assignments are a small set of partial assignments that collectively covers all assignments up to a given complete assignment (Definition 10).

   (b) **Extended Frege (eFrege) certification of the cumulator** (Section 4.2). Informally, the properties of restriction imply that after restricting a proof with a complete assignment α, the model count will be $\mathbf{0}$ or $\mathbf{1}$, depending on whether α satisfies the formula. Whereas, in the case of restricting a proof with a partial assignment α undefined on some variable x, the model count returned should be the sum of model counts returned when restricting with α_0 and α_1, where $\alpha_b = \alpha \cup \{x = b\}$. These two are the only properties we need to know which when globally combined tells us that the model count under the restriction of a partial assignment is correct. If a #SAT proof system admits easy eFrege proofs for these properties of restriction (Definition 19), we show that it can be p-simulated by CLIP+eFrege (Theorem 23).

   (c) **CLIP+eFrege p-simulates CPOG** (Section 5.3, 6). Using the structure of the POG to form an inductive proof. We explicitly show that the CPOG proof system admits easy eFrege proofs of the properties of restriction (Lemma 52). Thereby, proving that CLIP+eFrege p-simulates CPOG (Theorem 53), which in-turn p-simulates KCPS(#SAT) and MICE [3]. This shows that CLIP+eFrege p-simulates all existing #SAT proof systems.

3. **Upperbounds in CLIP for some hard formulas of existing #SAT systems**:

   (a) **XOR-PAIRS**. Since the CLIP framework uses unrestricted Boolean circuits in its proof as compared to other existing #SAT proof systems, CLIP is capable of handling formulas that are hard for other systems. We show this for the family XOR-PAIRS, which are known to be hard for MICE [5], and give an easy proof for the same in the CLIP+eFrege proof system (Theorem 65). For the short proof, we first carefully define a short cumulator for the XOR-PAIRS. Then, using a constant case analysis we certify the correctness of the cumulator in eFrege.

   (b) **Unsatisfiable formulas**. We show that any unsatisfiable formula which has an easy eFrege proof, also has an easy CLIP+eFrege proof (Proposition 55). It is already known that for unsatisfiable formulas, MICE and KCPS(#SAT) are p-equivalent to Resolution and regular-Resolution respectively [3, Proposition 5.1, 5.3]. As a result, all unsatisfiable formulas, which are hard for Resolution and easy for eFrege are all hard for MICE and KCPS(#SAT) but easy for CLIP+eFrege. We list two such important counting based unsatisfiable formulas. Namely, the pigeonhole principle (PHP) and the clique-coloring

principle [28, Definition 7.1], which are known to be hard for Resolution [24, 28] but are easy for eFrege [18, 9, 10].

Organization of the paper: Section 2 contains the preliminaries used in this paper. Section 3 defines the CLIP framework proof systems. We give the CLIP simulation technique Part-1 and Part-2 in Section 4.1 and 4.2 respectively. Section 5.1, 5.2 and 5.3 show Part-1 of the simulation technique for proof systems KCPS(#SAT), MICE′ and CPOG respectively. We show Part-2 of our simulation technique for CPOG proof system in Section 6. Section 7 shows upper bounds in the clip framework for hard formulas of MICE and KCPS(#SAT). Section 8 discusses the potential for new benchmarks in the SAT and DQBF worlds from the clip framework. Finally in Section 9, we conclude the paper.

2 Preliminaries

For a Boolean variable x, its literals can be x or $\neg x$. We use the notation $\overline{\ell} = \neg x$ when $\ell = x$ and $\overline{\ell} = x$ when $\ell = \neg x$. A clause C is a disjunction of literals and a conjunctive normal form (CNF) formula ϕ is a conjunction of clauses. We denote the empty clause by $\bot$. $\mathit{vars}(\phi)$ is the set of all variables in formula ϕ.

2.1 Assignments

A partial assignment is a partial mapping from a set of propositional variables X to $\{0, 1\}$, when the mapping is defined everywhere we say the assignment is complete. $\langle X \rangle$ is the set of all complete assignments. Consider a totally ordered set of variables X. An initial assignment α is a partial assignment to X such that there are no pairs $x, y \in X$ where $x < y$ and x is undefined in α and y is defined. $\mathit{vars}(\alpha)$ are the variables for which α is defined and $|\alpha|$ represents $|\mathit{vars}(\alpha)|$. A partial assignment α can be extended to a total assignment by appending 0/1 assignment to the variables $X \setminus \mathit{vars}(\alpha)$. Two partial assignments α, β are called non-overlapping, if there does not exist any total assignment γ which can be obtained by extending both α and β.

For a CNF ϕ, $\phi|_\alpha$ (similarly $C|_\alpha$) denotes the restricted formula (or clause) resulting from replacing all occurrences of $\mathit{vars}(\alpha)$ in ϕ (or C) with assignments from α. For a propositional formula F we define the indicator function $\mathbb{1}_F$, this acts on the free variables of F. $\mathbb{1}_F$ is equal to an assignment that corresponds to $\mathbf{0}$ when F is false and $\mathbf{1}$ when F is true.

When variables are ordered as $X = \{x_{n-1}, \ldots x_0\}$, complete assignments can be seen as binary numbers i.e. $\{x_2 = 1, x_1 = 0, x_0 = 1\}$ represents $\mathbf{5}$. We distinguish numerals $\mathbf{0}$ and $\mathbf{1}$ from Boolean constants 0 and 1 through the use of boldface. Let num map assignments to integers using the standard binary encoding ($\mathit{num}(\alpha) = \sum_{i=0}^{i<n} \mathbb{1}_{\alpha(x_i)} \cdot \mathbf{2}^i$), and num^{-1} be its inverse. We also encapsulate arithmetic statements with $||\cdot||$ to indicate that we revert this into a proposition. Later we will drop this notation when obvious. We denote $[J]$ to denote numbers $\{\mathbf{1}, \mathbf{2}, \ldots, J_1, J\}$ and $[J_1, J_2]$ to denote numbers $\{J_1, J_1 + \mathbf{1}, \ldots, J_2 - \mathbf{1}, J_2\}$.

2.1.1 Model Counting

Given a formula ϕ over a set of variables X, a model is an complete assignment to X that satisfies ϕ. The set of models of ϕ is $\mathcal{M}(\phi)$. We denote the total number of models of a CNF ϕ as $\#_{\text{models}}(\phi) = |\mathcal{M}(\phi)|$. #SAT is the computational problem of calculating $\#_{\text{models}}(\phi)$ from a CNF ϕ. The class of languages decidable in polynomial time with an oracle to #SAT are denoted by $P^{\#P}$.

The weighted model counting problem assigns a real-valued weight function $w : X \to [0, 1]$ to the set of variables X. The weighted counting value is $\#_w(\phi) = \sum_{\alpha \in \mathcal{M}(\phi)} \prod_{x \in X}^{\alpha(x)=1} w(x) \cdot \prod_{x \in X}^{\alpha(x)=0} (1 - w(x))$.

2.2 Circuits

A Boolean *circuit* σ on variables X is a directed acyclic graph, in which the input nodes (with in-degree 0) are Boolean variables in X and other nodes are the basic Boolean operations: $\vee$ (OR), $\wedge$ (AND) and $\neg$ (NOT) and have in-degree at most 2. Every Boolean circuit σ evaluates a Boolean function whose output is that of the node with out-degree 0 in σ. $\mathsf{P/poly}$ is the class of Boolean functions computed by polynomial-sized circuit families.

We refer to a *multi-circuit* when we have multiple nodes with out-degree 0. This is simultaneously many overlapping circuits. Multi-circuits take inputs and outputs of Boolean vectors of fixed length. We denote the Boolean XOR gate with $\oplus$ in the paper. Likewise we use $\leftrightarrow$ (or $=$, or $\equiv$) for bi-equivalence and $A \models B$ to mean that models of A are also models of B.

The class of polynomials with polynomial degree that can be represented by arithmetic circuits of polynomial size are represented by VP. VNP is the class of polynomials $f(x_1, ..., x_n)$ such that given a monomial, its coefficient in f can be determined efficiently with a polynomial size arithmetic circuit.

A Conjunctive Normal Form (CNF) ϕ can trivially be represented as a Boolean circuit σ as follows: for every $C \in \phi$, σ has $|vars(C)|$ number of OR-gates. Then, σ has $m-1$ AND-gates where m is the number of clauses $\in \phi$.

2.3 Proof Systems

A *proof system* [17] is a polynomial-time function that maps proofs to theorems, where the set of theorems is some fixed language L. A proof system is sound if its image is contained in L and complete if L is contained in its image. A proof system takes in strings as its inputs. Let π be such a proof we denote its *size*, i.e. the string length by $|\pi|$. Given two proof systems f and g for the same language L. We that the f *p-simulates* g, when there is a polynomial time function r that maps g-proofs to f-proofs such that $g(\pi_1) = f(r(\pi_1))$. f and g are said to be *p-equivalent* if they both p-simulate each other,f and g are incomaparable if neither of them p-simulates the other. We say that f is exponentially stronger than g, if f p-simulates g but g does not p-simulate f.

Conventionally we may take L to be the set of propositional tautologies (as in Section 2.3.1). For propositional model counting, we take L as the set of all pairs $(\phi, \#_{\text{models}}(\phi))$, where ϕ is any propositional formula. We refer to a #SAT proof of $(\phi, \#_{\text{models}}(\phi))$ as a proof of ϕ.

Definition 1 (Closure under restrictions [29])**.** *A proof system P is closed under restrictions if for every P-proof π of a CNF formula ϕ and any partial assignment α to $vars(\phi)$, there exists a P-proof π' of $\phi|_\alpha$ such that $|\pi'| \leq p(|\pi|)$ for some polynomial p. In addition, there exists a polynomial time procedure (w.r.t. $|\pi|$) to extract π' from π.*

A similar definition called 'closure under conditioning' exists in the knowledge-compilation domain [19, Definition 3]. Precisely, a knowledge representation structure S (like DNNF, POG, etc) is closed under conditioning if from an S structure T and an assignment α to $vars(T)$, another S structure T' can be computed just by replacing all occurrences of free variables by α wherever defined. Additionally T' should be equivalent to $T \wedge \alpha$.

2.3.1 Propositional Proof Systems

Resolution [32] is arguably the most studied propositional proof system. It has the following rule: $\frac{(C \vee x) \quad (D \vee \overline{x})}{(C \cup D)}$ where C, D are clauses and x is a variable. Resolution refutation ρ of CNF ϕ is a derivation of $\bot$ using the above rule (i.e. $\rho := \phi \vdash_{Res} \bot$). Size of ρ (i.e. $|\rho|$) is the number of times the above rule is used in ρ. It is well-known that Resolution is closed under restrictions.

Frege systems [22] are important propositional proof systems. They consist of a sound and complete set of axioms and rules where any variable can be substituted by any formula. All Frege systems are p-equivalent [17]. Figure 2 gives one example of a Frege system.

$$\frac{}{1} \qquad \frac{}{x_1 \to (x_2 \to x_1)} \qquad \frac{}{((x_1 \to 0) \to 0) \to x_1}$$

$$\frac{}{(x_1 \to (x_2 \to x_3)) \to ((x_1 \to x_2) \to (x_1 \to x_3))} \qquad \frac{x_1 \qquad x_1 \to x_2}{x_2}$$

Figure 2: A Frege system for connectives $\to, 0, 1$.

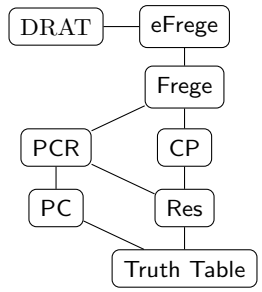


Figure 3: p-simulations of propositional proof systems [17].

Extended Frege (eFrege) [17] allows the introduction of new variables as well as all Frege rules. Simultaneously we can imagine it as a Frege system where lines are circuits instead of formulas, or as Substitution Frege, where derived tautologies can be generalised. eFrege is also p-equivalent to **DRAT** (Deletion Resolution Asymmetric Tautology) [27], a practical proof-format widely used in certifying SAT solvers.

In Figure 3 we show that eFrege sits at the top of the simulation hierarchy of propositional proof systems. In fact eFrege can simulate any proof system as long as there is a short proof of the reflection principle of said proof system [26].

When showing that eFrege has short proofs for complicated tautologies, it does not help us to be committed to one strictly defined proof system. Instead, we can use the fact that it can simulate many different proof systems such as Resolution, Cutting planes, Polynomial calculus and Truth Tables. Any tautology that has a short proof in any weaker system will also have a short proof in eFrege.

An NP-oracle is a theoretical concept which answers the membership question for NP in one step i.e if a formula Φ is given, it can correctly return if Φ belongs to SAT or not. NP-oracles have been used in QBF proof systems to skip propositional inference steps [13, 4].

3 Circuit Linear Induction Proposition (CLIP) Proof Framework

In this section, we define a propositional model counting proof framework (CLIP+$\mathcal{P}$) for any propositional proof system $\mathcal{P}$. Given a CNF ϕ over n variables, a CLIP+$\mathcal{P}$ proof consists of a Boolean circuit ξ (denoted as a *cumulator*) which outputs the total number of models of ϕ from complete assignment $\mathbf{0}$ to a given complete assignment $num(\alpha)$ (denoted as $C_{models}(\phi, \alpha)$, Definition 2). Clearly, when $num(\alpha) = \mathbf{2}^n - \mathbf{1}$, $C_{models}(\phi, \alpha) = \#_{\text{models}}(\phi)$. In addition, the CLIP+$\mathcal{P}$-proof also requires a $\mathcal{P}$-proof of a statement which carefully encodes the correctness of cumulator ξ using the induction-principle (see Definition 5). We need the following definitions.

Definition 2 ($C_{models}(\phi, \alpha)$)**.** *Let ϕ be an CNF formula, fix an order among vars(ϕ). For any complete assignment α to vars(ϕ), the cumulative number of models of CNF ϕ w.r.t. α (denoted by $C_{models}(\phi, \alpha)$) is the number of models of ϕ between assignment $\mathbf{0}$ to assignment num(α). In other words $C_{models}(\phi, \alpha) := \Sigma_{num(\beta) \leq num(\alpha)} \mathbb{1}_{\phi(\beta)}$, where $\mathbb{1}_{\phi(\beta)}$ is the indicator function for when β is a model of ϕ.*

Definition 3 (Cumulator)**.** *A cumulator for a CNF ϕ over n variables is a multi-circuit $\xi(\alpha)$ which takes as input a complete assignment α to vars(ϕ) (as n binary bits) and calculates the cumulative number of models of ϕ, i.e. $C_{models}(\phi, \alpha)$ outputted as $n+1$ binary bits. As a result, when α is the last assignment (i.e. $num(\alpha) = \mathbf{2^n - 1}$), $\xi(\alpha)$ outputs the total number of models of ϕ, we denote this as the final output of ξ.*

A trivial cumulator for ϕ would be to keep a counter and given any α, input every assignment from $\mathbf{0}$ to $num(\alpha)$ into the trivial Boolean circuit representing ϕ. If an assignment is a model then increment the counter. This will take $\mathcal{O}(2^{|vars(\phi)|})$ computations in the case of α being the last assignment.

Consider a CNF ϕ and let k be its number of models. Given a cumulator $\xi(\alpha)$ for ϕ, the correctness of the cumulator can be encoded inductively as follows:

For the base case when $num(\alpha) = \mathbf{0}$, we need to verify that the following is satisfied: $(\phi(\alpha) \wedge ||\xi(\alpha) = \mathbf{1}||) \vee (\overline{\phi(\alpha)} \wedge ||\xi(\alpha) = \mathbf{0}||)$. This covers the case that if the first assignment is a model for ϕ then the cumulator should return $\mathbf{1}$, else a $\mathbf{0}$.

For the inductive step when $num(\alpha) = num(\beta) + \mathbf{1}$, the following should be satisfied $(\phi(\alpha) \wedge ||\xi(\alpha) = \xi(\beta) + \mathbf{1}||) \vee (\overline{\phi(\alpha)} \wedge ||\xi(\alpha) = \xi(\beta)||)$. This covers the case that if the next assignment α after β is a model of ϕ, then the cumulator should increment its output by 1. Otherwise, the cumulator should output the same number under both assignments.

For the final case when $num(\alpha) = \mathbf{2}^{|vars(\phi)|} - \mathbf{1}$, it should be true that $\xi(x) = k$. This covers the case that the cumulator computes the correct total number of models of ϕ.

It is clear to see that if all of the above cases are true, the cumulator ξ is proven to be a correct cumulator of ϕ. From the above discussion, one can encode the correctness of ξ as the following statement ($||\cdot||$ encloses the arithmetic comparisons needed):
$||num(\alpha) = \mathbf{0}|| \rightarrow \big((\phi(\alpha) \wedge ||\xi(\alpha) = \mathbf{1}||) \vee (\neg\phi(\alpha) \wedge ||\xi(\alpha) = \mathbf{0}||)\big)$
$\wedge ||num(\alpha) = num(\beta) + \mathbf{1}|| \rightarrow \big((\phi(\alpha) \wedge ||\xi(\alpha) = \xi(\beta) + \mathbf{1}||) \vee (\neg\phi(\alpha) \wedge ||\xi(\alpha) = \xi(\beta)||)\big)$
$\wedge ||num(\alpha) = \mathbf{2}^{|vars(\phi)|} - \mathbf{1}|| \rightarrow ||\xi(x) = k||$.

To convert this into a purely propositional statement, we need Boolean circuits to implement the arithmetic conditions $||x = y||$ and $||x = y + \mathbf{1}||$ for any integers x, y. We define polynomial sized Boolean circuits for the same as $E(x, y)$ and $T(x, y)$ respectively in Definition 4 below.

Definition 4. *Let Z be a set of variables of size n, and let γ and δ be assignments to Z. For pairs of individual variables a, b, use $a = b$ to denote $(\neg a \vee b) \wedge (a \vee \neg b)$. We can encode polynomial size propositional circuits:*

- *$E(\gamma, \delta)$, that denotes $num(\gamma) = num(\delta)$: $E_0(\gamma, \delta) := (\gamma_0 \leftrightarrow \delta_0)$. For $1 \leq i < n$, $E_i(\gamma, \delta) := (\gamma_i \leftrightarrow \delta_i) \wedge E_{i-1}(\gamma, \delta)$. $E(\gamma, \delta) := E_{n-1}(\gamma, \delta)$.*
- *$T(\gamma, \delta)$, that denotes $num(\gamma) = num(\delta) + \mathbf{1}$ using an intermediate definition S that will denote the successor function. For $0 \leq i < n$ and accepting the empty conjunction as true, $S(\delta)_i := \neg(\delta_i \leftrightarrow \bigwedge_{j \geq 0}^{j<i} \delta_j)$. $T(\gamma, \delta) := E(\gamma, S(\delta)) \wedge \bigvee_{i \geq 0}^{i<n} \overline{\delta}_i$.*

Definition 5 (CLIP+$\mathcal{P}$)**.** *For every propositional proof system $\mathcal{P}$, the* CLIP+$\mathcal{P}$ *system for #SAT is a cumulator ξ for a CNF ϕ along with its correctness presented as a valid $\mathcal{P}$-proof of the following linear induction proposition statement* lip(ξ)*.*

Let A and B be two disjoint copies of the variables in ϕ. The following is a tautology in the variables of $A \cup B$:

lip(ξ) :=

$E(A, num^{-1}(\mathbf{0})) \rightarrow \big((\overline{\phi(A)} \rightarrow E(\xi(A), num^{-1}(\mathbf{0}))) \wedge (\phi(A) \rightarrow T(\xi(A), num^{-1}(\mathbf{0})))\big) \wedge$

$T(B, A) \rightarrow \big((\overline{\phi(B)} \rightarrow E(\xi(B), \xi(A))) \wedge (\phi(B) \rightarrow T(\xi(B), \xi(A)))\big) \wedge$

$E(A, num^{-1}(\mathbf{2}^{|vars(\phi)|} - \mathbf{1})) \rightarrow E(\xi(A), num^{-1}(k))$.

The existence of a valid $\mathcal{P}$-proof of lip(ξ), ensures that ξ is correct and the final output k of ξ is the correct number of models of ϕ. Note that in a technical sense the proof of inductive step (i.e. line 2 in lip(ξ)) is sufficient to verify the cumulator ξ, as the base and final case can be managed in the checker.

Theorem 6. *If $\mathcal{P}$ is a propositional proof system then* CLIP+$\mathcal{P}$ *is a propositional model counting proof system.*

Proof. CLIP+$\mathcal{P}$ is sound and complete for #SAT as a trivial cumulator always exists for any ϕ and the propositional proof system $\mathcal{P}$ is sound and complete. Note that for a refutational proof system $\mathcal{P}'$, CLIP+$\mathcal{P}'$ can include the correctness of ξ by including a $\mathcal{P}'$-refutation of $\overline{\text{lip}(\xi)}$ from the above definition. For polynomial time checkability, we perform 3 steps: 1) Verify that ξ is indeed a circuit. 2) Using ξ, generate lip(ξ) once again, to make sure it matches (where $\mathcal{P}$ does not accept circuits a canonical translation, i.e. a Tseitin transformation is needed). 3) Verifying the $\mathcal{P}$ proof. □

Theorem 7. CLIP+eFrege *has a super-polynomial lower bound only if* eFrege *has a super-polynomial lower bound or* $\mathsf{P}^{\#\mathsf{P}} \nsubseteq \mathsf{P/poly}$.

Proof. Suppose there is a family $(\phi_n)_{n\geq 0}$ of propositional formulas that are a super-polynomial lower bound to CLIP+eFrege. Let $f_{n,i}$ be the i^{th} bit of the cumulative function for ϕ_n. $(f_{n,i})_{n\geq 0}^{0\leq i\leq |vars(\phi_n)|}$ is a $\mathsf{P}^{\#\mathsf{P}}$ family. Finding the value of the cumulator at assignment α can be found by adding a constraint to ϕ that the only acceptable models are less than or equal to α and querying for the number of models.

Now suppose $\mathsf{P}^{\#\mathsf{P}} \subset \mathsf{P/poly}$, then there are polynomial size circuits for each $f_{n,i}$ and thus a polynomial size cumulator ξ_n for each ϕ_n. For each n, lip(ξ_n) is also polynomial size in ϕ_n. Thus the family $(\text{lip}(\xi_n))_{n\geq 0}$ is super-polynomial lower bound for eFrege. □

Definition 8 (CLIP^{NP})**.** *The proof format* CLIP^{NP} *for #SAT is a cumulator ξ for CNF Φ along with its correctness encoded by the statement* lip(ξ) *from Definition 5 and verified by an NP-oracle instead of a proof.*

For CLIP$^{\mathsf{NP}}$, size is determined by lip(ξ) only. CLIP^{NP} is a proof system only if $\mathsf{P} = \mathsf{NP}$.

4 CLIP+eFrege simulates existing #SAT proof systems

In this section, we give an important CLIP+eFrege p-simulation technique for any #SAT proof systems which are closed under restrictions and have short eFrege proofs of the properties of restriction (Definition 19). To be precise, we show that CLIP+eFrege can p-simulate any model counting proof system $\mathcal{P}$ which obey the following conditions:

I. The polynomial-time ability to extract circuits from a $\mathcal{P}$-proof π of CNF ϕ over n variables $(x_{n-1} \dots x_0)$ that calculate closure under restrictions for any given (partial) assignment α. We denote this circuit as $\theta^i(\alpha)$, where $i = n - |\alpha|$. Precisely, $\theta^i(\alpha)$ computes the count of models of ϕ that also satisfy α (Definition 9).

II. $\mathcal{P}$ has short eFrege proofs for properties of restriction (Definition 19) that confirm the correctness of closure under restriction in $\mathcal{P}$. Precisely, the properties encode that $\theta^0(\alpha) = \phi(\alpha)$ and $\theta^i(\alpha) = \theta^{i-1}(\alpha \cup \{x_{i-1} = 1\}) + \theta^{i-1}(\alpha \cup \{x_{i-1} = 0\})$ when $i > 0$.

Let us formally define the circuit θ used in above conditions.

Definition 9. *Let ϕ be a CNF on n variables and α be a (partial) assignment of length $n - i$. We define $\theta^i(\alpha)$ to be a Boolean circuit that returns $\#_{models}(\phi|_\alpha)$. Also, $\theta^i_j(\alpha)$ is a circuit that returns the j^{th} bit of $\theta^i(\alpha)$.*

In the upcoming subsections, we give the complete simulation technique. Recall that CLIP+eFrege proof consists of a cumulator ξ and a eFrege-proof of validity of the propositional 'lip' statement which encodes the correctness of ξ. Using the condition-I above, in Section 4.1 we derive the cumulator ξ for ϕ (Part 1 of our simulation technique). In Section 4.2, we use the condition-II to derive the eFrege-proof of lip(ξ) (Part 2 of our simulation technique).

4.1 Simulation Technique (Part 1) : Cumulator Extraction

In this section, we give a general framework of extracting efficiently a cumulator from the proofs of existing propositional model counting proof systems. We need the following definition.

Definition 10 (Disjoint binary partial assignment cover (PAC(J_1,J_2)))**.** *Let J_1, J_2 be integers representing some complete assignments to variables $X := x_{n-1}, ..., x_0$ in this order. The cover PAC(J_1,J_2) is a set of partial assignments to X which are non-overlapping and together cover the entire assignment space between J_1 and J_2 inclusive of both.*

For instance, let $X := \{x_3, x_2, x_1, x_0\}$, $J_1 := \mathbf{5}$ and $J_2 := \mathbf{15}$. One possible PAC(J_1,J_2):= $\{\{x_3 = 1\}, \{x_3 = 0, x_2 = 1, x_1 = 1\}, \{x_3 = 0, x_2 = 1, x_1 = 0, x_0 = 1\}\}$. Observe that the first partial assignment (i.e. $\{x_3 = 1\}$) is covering all assignments from $[\boldsymbol{8}, \boldsymbol{15}]$. Similarly the second and third partial assignments are covering the assignments $[\boldsymbol{6}, \boldsymbol{7}]$ and $\mathbf{5}$ respectively.

Let us now outline the general extraction technique.

Cumulator Extraction Technique: Let $\mathcal{P} \in \{\mathsf{MICE}, \mathsf{KCPS}(\#\mathsf{SAT}), \mathsf{CPOG}\}$ be a propositional model counting proof system. Consider a CNF ϕ over n variables and its $\mathcal{P}$-proof π. In order to efficiently extract a correct cumulator ξ for ϕ, we follow the following steps:

1. Show that $\mathcal{P}$ is closed under restrictions (see Definition 1). That is, show that $\mathcal{P}$ obeys condition-I from above.

2. For any complete assignment J to *vars*(ϕ), find the set of non-overlapping partial assignments (to *vars*(ϕ)) which cover the entire assignment space from assignments $0 \to J$ (i.e PAC(0,J) see Definition 10).

Using Fenwick's idea [20], it is easy to compute PAC(0,J) for any complete assignment J (Lemma 11). Moreover, $|\mathsf{PAC}(0,J)| \leq n$.

3. For each partial assignment $\alpha \in$ PAC(0,J), restrict π with α and consider the $\mathcal{P}$-proof π' of CNF $\phi|_\alpha$. Observe that π' is actually $\theta^i(\alpha)$ where $i = n - |\alpha|$. Since $\mathcal{P}$ is closed under restrictions, this step takes $\mathcal{O}(|\pi|)$ time for every α.

4. Finally add the number of models returned by all the π' proofs obtained in the above step. (This step will need a full-adder circuit as integers are represented as $(n+1)$-bit numbers).

This process will return $\xi(J)$ which computes $C_{models}(\phi, J)$ and takes $\mathcal{O}(n.|\pi|)$ time.

We prove Step-1 of our simulation technique individually for existing proof systems KCPS(#SAT), MICE′ and CPOG in Section 5.1, 5.2 and 5.3 respectively. For Step-2, consider the following lemma.

Lemma 11. *Given an input size n, and binary integer $0 \leq J < 2^n$. There is a polynomial time algorithm in n that returns a disjoint binary partial assignment cover for $[\boldsymbol{0}, J]$ (PAC(0,J)) with at most n many partial assignments.*

Proof. Refer to Algorithm 1 for the exact procedure. The idea is as follows: given any total assignment J, it finds the index of the least significant 1 (from the right) in its binary representation (say r) and subtracts $(2^r + 1)$ from J to find its parent assignment. The process repeats recursively for the parent assignment while it remains > 0. However all these parent assignments are total. In order to find the required partial assignments, we simultaneously remember (in the dash array) logarithm of the difference between any assignment and its parent. As a result, the dash array records the number of variables (from the right end) that need to be removed from the corresponding total assignment.

The correctness of Algorithm 1 stems from the correctness of Fenwick trees [20] which efficiently computes the cumulative sum of numbers stored between index 0 to any other index of the array by visiting least possible indices in the tree. A Fenwick tree guarantees that it visits logarithmic number of indexes in the worst case. Algorithm 1 similarly computes PAC(0,J) by finding the least number of partial assignments to include such that they cover the entire range from $[0, J]$. Hence the maximum partial assignments returned from Algorithm 1 is $log(2^n) = n$. Note that we drop the use of a sign bit as we only deal with non-negative numbers for assignments in Fenwick trees. □

Algorithm 1 Fenwick tree [20] based algorithm to find PAC(0,J)

Require: $J < 2^n$
  function FENWICK-ASSIGNMENTS(int J, int n)
    int $\alpha := \{\}$, dash:= $\{\}$ /*α, dash are each a set of integers*/
    int $indx \leftarrow J + 1$ /*assignments $\in [0, 2^n - 1]$ but the Fenwick tree handles $[1, 2^n]$*/
    while $indx > 0$ **do**
      parent = $indx - (indx \ \& \ -indx)$ /*& is the bit-wise AND operator*/
      α.append(parent)
      dash.append($log(indx -$ parent)) /*records the no. of variables to forget from α*/
      $indx \leftarrow$ parent
    end while
  return $\alpha' =$ process(α, dash)
  /*'process' function does the following: for $i \in |\alpha|$, $\alpha'[i]$ is the partial assignment obtained from $\alpha[i]$ after discarding 'dash[i]' number of variables from the right end in the fixed ordering of variables*/
  end function

For a detailed version of Algorithm 1, see Appendix A. We provide Example 12 to illustrate the use of Algorithm 1.

Example 12. *Let $n = 5$ and $J = 22$. Let the variables be lexicographical ordered as $x_4 \dots x_0$. The Algorithm 1 will initially find the parent of $indx = 23$, this is done as follows:*

$indx = 10111$, $-indx = 01001$

$\implies (indx \ \& \ -indx) = 00001 \implies indx - (indx \ \& \ -indx) = 10110 = (22)_{10}$

Doing this recursively gives the parent chain as $22 \to 20 \to 16 \to 0$.

The sets α and dash are $\{22, 20, 16, 0\}$ and $\{0, 1, 2, 4\}$ respectively. The corresponding PAC(0,22) is the following set of partial assignments: $\Big\{\{x_4 = 1, x_3 = 0, x_2 = 1, x_1 = 1, x_0 = 0\},\ \{x_4 = 1, x_3 = 0, x_2 = 1, x_1 = 0\},\ \{x_4 = 1, x_3 = 0, x_2 = 0\},\ \{x_4 = 0\}\Big\}$.

Note that extracting cumulator is not enough for the full CLIP simulation. Because, CLIP proofs also consists of the validity proof of the lip statement. However, with an access to an NP-oracle, the validity of the lip statement can be obtained in one step due to the correctness of our simulation technique. Thus the efficient cumulator extraction shows that CLIP^{NP} (Definition 8) p-simulates any #SAT system which is closed under restrictions. Observe that CLIP^{NP} is a proof system only if P = NP.

In the upcoming sections, we give full CLIP framework simulations of all the existing #SAT proof systems using the powerful eFrege system for validating the lip statement.

Recall that for the cumulator extraction, we used the concepts of Fenwick assignments. In upcoming proofs, we also need some additional results on Fenwick assignments. We finish this subsection with these results before proceeding to the part 2 of our simulation technique.

4.1.1 Formalising Fenwick Assignments

In Lemma 11, we show how to compute the PAC given a complete assignment α with $|\mathsf{PAC}(0,\alpha)| \leq |\alpha|$. In this section, we formalise it as a Boolean circuit (Definition 13) which outputs a PAC for any given complete assignment α. We call the output $\mathsf{PAC}(0,J)$ of the Boolean circuit as Fenwick assignments. Note that these are the same assignments obtained when following Algorithm 1. We further show that eFrege can handle a few essential properties of Fenwick assignments. We use these in the next section for part-2 of the simulation technique.

Definition 13 (Fenwick assignments)**.** *Let α be a complete assignment to n variables with ordering $x_{n-1} \dots x_0$. For every $i, 0 \leq i \leq n$, we define an existence function $e_i(\alpha)$ and the set of initial assignment bits $f_{i,j}$ for $0 \leq i \leq j < n$ as follows:*

$$e_i(\alpha) = \begin{cases} 1 & \alpha(x_i) \wedge \bigvee_{k\geq 0}^{k<i} \overline{\alpha(x_k)} \text{ and } 0 < i < n \\ 1 & \overline{\alpha(x_i)} \wedge \bigwedge_{k\geq 0}^{k<i} \alpha(x_k) \text{ and } 0 \leq i < n \\ 1 & \bigwedge_{k\geq 0}^{k<n} \alpha(x_k) \text{ and } i = n \\ 0 & \text{otherwise} \end{cases} \qquad f_{i,j}(\alpha) = \begin{cases} 1 & \alpha(x_j) \text{ and } j > i \\ 0 & \text{otherwise} \end{cases}$$

We denote for $0 \leq i < n$, $f_i(\alpha) = \{f_{i,j} | i \leq j < n\}$ as the i^{th} partial assignment for α (note that f_n is the empty assignment and needs no variables to be defined). For a complete assignment α, Fenwick assignments are $\{f_i(\alpha) | e_i(\alpha) = 1, 0 \leq i \leq n\}$. Here, $e_i(\alpha)$ can be seen as a single-bit value that indicates if there is an initial assignment defined on $n-i$ variables in the Fenwick assignments of α. Similarly, $f_{i,j}(\alpha)$ is the value of x_j in the i^{th} partial assignment corresponding to $e_i(\alpha)$ in Fenwick assignments of α.

Below we give an example of how we represent Fenwick assignments as in Definition 13.

Example 14. *Let $n = 4$ and $J = \mathbf{12}$. Let the variables be lexicographical ordered as $x_3 \dots x_0$. That is, $\alpha := \{x_3 = 1, x_2 = 1, x_1 = 0, x_0 = 0\}$.*

*The Fenwick indicators have the following values: $e_0(\mathbf{12}) = 1$, $e_1(\mathbf{12}) = 0$, $e_2(\mathbf{12}) = 1$, $e_3(\mathbf{12}) = 1$, $e_4(\mathbf{12}) = 0$. This indicates that there are 3 partial Fenwick assignments in PAC(**0**,**12**) ending at x_0, x_2 and x_3 respectively. The exact assignments are computed as follows:*
$f_{0,3}(\mathbf{12}) = 1,\ f_{0,2}(\mathbf{12}) = 1,\ f_{0,1}(\mathbf{12}) = 0,\ f_{0,0}(\mathbf{12}) = 0 \quad \to f_0 = \{x_3 = 1, x_2 = 1, x_1 = 0, x_0 = 0\}$
$f_{2,3}(\mathbf{12}) = 1,\ f_{2,2}(\mathbf{12}) = 0 \quad \to f_2 = \{x_3 = 1, x_2 = 0\}$
$f_{3,3}(\mathbf{12}) = 0 \quad \to f_3 = \{x_3 = 0\}$.

*The corresponding PAC(**0**,**12**) (i.e Fenwick assignment of **12**) is the following set of partial assignments: $\Big\{\{x_3 = 1, x_2 = 1, x_1 = 0, x_0 = 0\},\ \{x_3 = 1, x_2 = 0\},\ \{x_3 = 0\}\Big\}$. Note that these can also be obtained from Algorithm 1 (see Example 12).*

Next we see few properties of how Fenwick assignments change as $num(\alpha)$ increases slowly. Assume X, Y are complete assignments and $Y = X + 1$. In the first case that X is odd and Y is even, the change in Fenwick assignments are minor and easily captured in eFrege.

Lemma 15. *Assume $T(Y, X)$ (i.e. $Y = X + 1$) and $\neg Y_0$ then*

1. *$e_0(Y) \wedge \neg e_0(X)$*
2. *For $i > 0$, $e_i(X) \leftrightarrow e_i(Y)$*
3. *For $0 < i \leq j < n$, $e_i(X) \to (f_{i,j}(X) \leftrightarrow f_{i,j}(Y))$*
4. *For $0 \leq j < n$, $f_{0,j}(Y) \leftrightarrow Y_j$*

And we can prove these formally in eFrege *in short proofs even in the case that Y and X are vectors of variables (or extension variables).*

Proof. $\bar{Y}_0 \wedge \bigwedge_{k\geq 0}^{k<0} Y_k$ is true hence $e_0(Y)$ can be shown via definition. Since, $Y_0 = X_0 \oplus \bigwedge_{k\geq 0}^{k<0} X_k$ and $\bigwedge_{k\geq 0}^{k<0} X_k$ is just the empty conjunction which equals 1 therefore X_0 is true and so $e_0(X)$ must be false and we can show this through a short derivation.

Take p to be the maximum such that $\bigwedge_{k\geq 0}^{k<p} X_i$ is true, we can prove such a maximum exists by exhibiting a disjunction. Then for $0 < i < p$, $e_i(X) = 0$. $Y_k = 0$ for $k \leq i$ by definition of T and so $\neg e_i(Y)$ by definition of $e_i(Y)$. For $i = p$, $X_i = 0$ while $\bigwedge_{k\geq 0}^{k<i} X_i$ and $Y_i = 1$ while $\bigvee_{k\geq 0}^{k<i} \bar{Y}_i$ so both $e_i(X)$ and $e_i(Y)$ are true. $f_{p,p}(X) = f_{p,p}(Y) = 0$ because i is not strictly greater than itself. For $j > p$, we have to show that X_j and Y_j are equal. Recall that $Y_j = X_j \oplus \bigwedge_{k\geq 0}^{k<j} X_k$, but since X_p is false, $\bigwedge_{k\geq 0}^{k<j} X_k = 0$ and so $Y_j = X_j$. Hence $f_{p,j}(X) = f_{p,j}(Y)$.

For $i > p$, if $e_i(X)$ is true then $X_i \wedge \bigvee_{k\geq 0}^{k<i} \bar{X}_k$ must be true. $\bigvee_{k\geq 0}^{k<i} \bar{Y}_k$ must also be true because Y_0 is true. Since $X_i = Y_i$ then $Y_i \wedge \bigvee_{k\geq 0}^{k<i} \bar{Y}_k$ is also true and so $e_i(Y)$ can be proven that way. Since $X_j = Y_j$ for $j \geq i$ then by definition $f_{i,j}(X) = f_{i,j}(Y)$.

By definition, for $j > 0$, $f_{0,j}(Y) = Y_j$ and $f_{0,0}(Y) = 0 = Y_0$.

□

In the second case that X is even and Y is odd, the change in Fenwick assignments maintain some essential properties which are again easy to capture in eFrege.

Lemma 16. *Assume $T(Y, X)$ (i.e. $Y = X + 1$) and Y_0 then there is some maximum $p : 0 < p \leq n$ such that $\bigwedge_{j\geq 0}^{j<p} Y_j$. Further the following properties are true and have short formal eFrege proofs.*

1. *(a) $e_i(X) \wedge \neg e_i(Y)$ for $0 \leq i < p$*

(b) $f_{i,j}(X) \leftrightarrow Y_j$ *for* $0 \leq i < p \leq j < n$

(c) $f_{i,j}(X)$ *for* $0 \leq i < j < p \leq n$

(d) $\neg f_{i,p}(X)$ *for* $0 \leq i < p$

2. (a) $\neg e_p(X) \wedge e_p(Y)$ *if* $p < n$

(b) $f_{p,j}(X) \leftrightarrow f_{p,j}(Y)$ *for* $p \leq j < n$.

(c) $\neg f_{p,p}(X) \wedge \neg f_{p,p}(Y)$ *if* $p < n$

3. (a) $e_i(X) \leftrightarrow e_i(Y)$ *for* $p < i < n$

(b) $f_{i,j}(X) \leftrightarrow f_{i,j}(Y)$ *for* $p < i \leq j < n$

Proof. Because $Y_0 = 1$, then by definition of T we can prove $X_0 = 0$, hence $\bigvee_{k\geq 0}^{k<i} \bar{X}_k$ is always true for $i > 0$. This means that through the definition of T, $Y_i = X_i$ for $i > 0$. This will be important for many items when $i > 0$.

For $i = 0$ we get $e_0(X)$ and $\neg e_0(Y)$ through definition and use of $Y_0 \wedge \neg X_0$. And for $0 < i < p$, $X_i = 1$ and since $\bigvee_{k\geq 0}^{k<i} \bar{X}_k$ is true $e_i(X)$ is true. However $\bigwedge_{j\geq 0}^{j\leq i} Y_j$ is true so $e_i(Y)$ is false. Through $Y_i = X_i$ we also get that for $0 \leq i < j$, $f_{i,j}(X) = X_j = Y_j$. For $j < p$ we specifically get $Y_j = 1$ and for $j = p$ we get $Y_j = 0$. This completes all cases from 1.

In case 2, $e_p(X)$ is false because $X_p = Y_p = 0$ and $\bigwedge_{k\geq 0}^{k<p} X_k$ is false. Likewise, $e_p(X)$ is true because $Y_p = 0$ and $\bigwedge_{k\geq 0}^{k<p} Y_k$ is true. $f_{p,p}(X) = f_{p,p}(Y) = 0$ by definition and $f_{p,j}(X) = f_{p,j}(Y)$, for $j > p$ because then $X_j = Y_j$.

In case 3, again $f_{i,j}(X) = f_{i,j}(Y)$, for $j \geq i > p$ because $X_j = Y_j$. We can also use $X_j = Y_j$ to show $e_i(X) = e_i(Y)$ because $\bigvee_{k\geq 0}^{k<i} \bar{X}_k$ and $\bigvee_{k\geq 0}^{k<i} \bar{Y}_k$ are now both true.

All these cases can be formalised in eFrege proofs due to their simplicity for each choice of p. One final important step is to create and prove disjunction over all possible p.

□

Step 3,4 of our simulation technique require restricting the θ circuit with all the Fenwick assignments of some complete assignment α and adding them up to get $\xi(\alpha)$. Using the formal definition of the Fenwick assignments from Definition 13, we have the following.

Definition 17. *For a complete assignment α on n variables, we define the vector of Boolean variables $\xi(\alpha)$ as the following sum:*

$$(e_n(\alpha) \wedge \theta^n(f_n(\alpha))) + (e_{n-1}(\alpha) \wedge \theta^{n-1}(f_{n-1}(\alpha))) + \cdots + (e_0(\alpha) \wedge \theta^0(f_0(\alpha)))$$

This circuit ξ is the required cumulator.

4.2 Simulation Technique (Part 2): eFrege certification of the cumulator

Recall that, for a full CLIP+eFrege simulation, the proof system must have short eFrege proofs of the properties of restriction, i.e. condition-II from Section 4. Let us formally define the properties of restriction in Definition 19 which are based on the following simple observations of partial assignments .

Observation 18. *Let ϕ be a CNF on variables $x_{n-1} \dots x_0$ (used in that lexicographic ordering in CLIP). Let α be a partial assignment defined on $x_{n-1} \dots x_i$, undefined on $x_{i-1} \dots x_0$. Given a $\{0,1\}$-value b, let $\alpha_b := \alpha \cup \{x_{i-1} = b\}$. Then, $\#_{models}(\phi|_\alpha) = \#_{models}(\phi|_{\alpha_0}) + \#_{models}(\phi|_{\alpha_1})$. If α is a complete assignment, $\#_{models}(\phi|_\alpha) = \mathbb{1}_\phi(\alpha)$.*

Proof. A complete assignment α either satisfies ϕ or falsifies it. Therefore $\#_{models}(\phi|_\alpha) = \mathbb{1}_\phi(\alpha)$. For a partial assignment α, $\#_{models}(\phi|_\alpha)$ calculates the models of ϕ after restricting with α. Notice that adding another restriction to $\phi|_\alpha$ only reduces the search space for models. However, adding the extra restriction in both the polarities again has the original search space for models as $\phi|_\alpha$. That is, $\#_{models}(\phi|_\alpha) = \#_{models}(\phi|_{\alpha_0}) + \#_{models}(\phi|_{\alpha_1})$. □

Definition 19 (Properties of Restriction)**.** *Let S be a propositional model counting proof system and ϕ be a CNF on n variables ($x_{n-1} \dots x_0$). Suppose ϕ has an S-proof π with θ being the associated circuit for restriction. We consider the following properties for all assignments α over $x_{n-1} \dots x_0$.*

1. *If α is a complete assignment: $\theta^0(\alpha) = \mathbb{1}_\phi(\alpha)$.*
2. *If α is a partial assignment defined on $x_{n-1} \dots x_i$: $\theta^i(\alpha) = \theta^{i-1}(\alpha_0) + \theta^{i-1}(\alpha_1)$*

Observation 20. *Any propositional model counting proof system $\mathcal{P}$ which is closed under restrictions, satisfies the properties of restriction mentioned in Definition 19.*

Proof. By definition, closure under restrictions imply the existence of a polynomial-size (in terms of the $\mathcal{P}$-proof of some CNF ϕ) Boolean circuit θ calculating the value of $\#_{models}(\phi|_\alpha)$ for any partial assignment α to $vars(\phi)$. This when combined with the properties of partial assignments shown in Observation 18, give the properties of restriction. □

Remark. *For the* CLIP+eFrege *simulation of a propositional model counting proof system, just satisfying the properties of restriction (Definition 19) are not enough. The proof system must have a short eFrege proof of these properties a well. In Lemma 52, we explicitly show that* CPOG *has short* eFrege *proofs for these properties.*

Next we prove in Lemma 21 that short eFrege proofs of the properties of restriction can be used to give short eFrege-proofs of the lip statement from Definition 5.

Before presenting the detailed proof of Lemma 21, we briefly give the proof idea. For a CNF ϕ and any two consecutive assignments β^1, β^2 such that $\beta^2 = \beta^1 + \mathbf{1}$, we need to show that $\xi(\beta^2) = \xi(\beta^1) + \mathbb{1}_\phi(\beta^2)$. We show this in the following two cases: β^2 being odd or even. In the case of β^1= odd and β^2= even, we show that the Fenwick assignments of β^2 are the Fenwick assignment of β^1 along with the assignment β^2 itself (Lemma 15). This directly implies $\xi(\beta^2) = \xi(\beta^1) + \mathbb{1}_\phi(\beta^2)$.

In the case of β^1= even and β^2= odd, we show that the only extra assignment in Fenwick assignments of β^2 is assignment γ which is the common prefix of both β^1 and β^2 (Lemma 16). We then show that using Observation 20 a linear number of times we can prove that $\theta(\gamma)$ decomposes so that it implies $\xi(\beta^2) = \xi(\beta^1) + \mathbb{1}_\phi(\beta^2)$.

Lemma 21. *Suppose $\mathcal{P}$ is a propositional model counting proof system which is closed under restrictions. Let ϕ be a CNF and ξ be a cumulator obtained by using Fenwick assignments on the $\mathcal{P}$-proof of ϕ. If $\mathcal{P}$ has polynomial-sized* eFrege *proofs of the properties of restriction, then it has short* eFrege *proof of lip(ξ).*

Proof. Line-1 of the lip statement assumes the assignment $\alpha = \mathbf{0}$. From a simple computation, only $e_0(\mathbf{0}) = 1$ with the corresponding $f_0(\mathbf{0}) = \mathbf{0}$. From Definition 17, $\xi(\mathbf{0}) = \theta^0(\mathbf{0})$. From the properties of restriction $\theta^0(\mathbf{0}) = \mathbb{1}_\phi(\mathbf{0})$.

Line-3 of the lip statement assumes the assignment $\alpha = \mathbf{2^n - 1}$. Similarly, we can compute that only $e_n(\mathbf{2^n - 1}) = 1$ and $f_n(\mathbf{2^n - 1}) = \emptyset$. Then $\xi(\mathbf{2^n - 1}) = \theta^n(\emptyset)$. θ^n contains no restriction, so it is the intended answer for the entire model count.
The main part of CLIP is the inductive step for complete assignments $\beta^2 = \beta^1 + \mathbf{1}$.

For an even β^2, We use the cases for short proofs in Lemma 16 . We can argue that

$$e_i(\beta^1) \leftrightarrow e_i(\beta^2) \text{ for } i > 0$$

and

$$e_i(\beta^1) \rightarrow (f_{i,j}(\beta^1) \leftrightarrow f_{i,j}(\beta^2)) \text{ for } i > 0.$$

That is, all Fenwick assignments for β^1 and β^2 are the same except the one corresponding to $e_0(\beta^2)$. Additionally that $f_0(\beta^2) = \beta_2$. Along with associativity (see Appendix B), this easily leads to $\xi(\beta^2) = \xi(\beta^1) + \mathbb{1}_\phi(\beta^2)$ in eFrege.

For an odd β^2, we can use the short proofs from Lemma 16 of the various cases . We take the maximum $p : 0 \leq p \leq n$ such that $\bigwedge_{j<p}^{j\geq 0} \beta_j^2$. We can derive (case-3a,3b of Lemma 16)

$$e_j(\beta^1) \leftrightarrow e_j(\beta^2) \text{ and } f_{j,k}(\beta^1) \leftrightarrow f_{j,k}(\beta^2) \text{ for } j > p.$$

At $j = p$, we have that (case-2a,2b of Lemma 16) $f_p(\beta^2) =$ the common prefix of β^1 and β^2 up to p (say γ). For $j < p$, we derive (case-1b of Lemma 16) that γ is the prefix of all these Fenwick assignments. Further we show (case-1c,1d of Lemma 16) that these assignments extend γ by $p-j-1$ number of 1s and end with a 0. That is, (extending the notation from Observation 18)

$$\{f_j(\beta^1)\}_{p>j\geq 0} = \gamma_0, \gamma_{10}, \gamma_{110}, \ldots, \beta^1.$$

Using the split property of restrictions for i times, we have

$$\theta(\gamma) = \sum_{j\leq p}^{j\geq 0} (e_j(\beta^1) \wedge \theta^j(f_j(\beta^1))) + \theta^0(\beta^2).$$

Adding $f_{j,k}(\beta^1) \leftrightarrow f_{j,k}(\beta^2)$ for $j > p$ to the above gives us $\xi(\beta^2) = \xi(\beta^1) + \mathbb{1}_\phi(\beta^2)$. □

Next, we give a supplementary example of Lemma 21.

Example 22. *Let a CNF ϕ be defined on $n = 5$ variables and β^1, β^2 be $10010, 10011$ (i.e* **18**, **19***) respectively. Let the variables be lexicographical ordered as $x_4 \ldots x_0$. The corresponding* PAC(**0**,**18**) *and* PAC(**0**,**19**) *are: $\Big\{\{x_4 = 1, x_3 = 0, x_2 = 0, x_1 = 1, x_0 = 0\},\ \{x_4 = 1, x_3 = 0, x_2 = 0, x_1 = 0\},\ \{x_4 = 0\}\Big\}$ and $\Big\{\{x_4 = 1, x_3 = 0, x_2 = 0\},\ \{x_4 = 0\}\Big\}$ respectively.*

The first zero of β^2 occurs for the second digit therefore we take $p = 2$. For $j > p$, clearly $e_4(\beta^1) = e_4(\beta^2)$ and $f_4(\beta^1) = f_4(\beta^2)$. There is one partial assignment that is in β^2 but not in β^1, it is $f_2(\beta^2)$ (i.e $\{x_4 = 1, x_3 = 0, x_2 = 0\}$). This is also the common prefix γ of both β^1 and β^2. Using the split property on γ twice, we have the following: $\theta^2(\gamma) = \theta^1(\gamma_0) + \theta^1(\gamma_1) = \theta^1(\gamma_0) + \theta^0(\gamma_{10}) + \theta^0(\gamma_{11})$. Adding $f_4(\beta^1) = f_4(\beta^2)$ to this implies that $\xi(\mathbf{19}) = \xi(\mathbf{18}) + \mathbb{1}_\phi(\mathbf{19})$.

This ends our simulation technique. For a model counting proof system $\mathcal{P}$ and a CNF ϕ with it's $\mathcal{P}$-proof π, we have a cumulator ξ from Part 1 of our simulation technique. In Lemma 21, we also have an eFrege proof of $\text{lip}(\xi)$. Therefore, we have the following.

Theorem 23. *CLIP+eFrege p-simulates any model counting proof system which is closed under restrictions and has short eFrege proofs of the properties of restriction.*

In conclusion, for any propositional model counting proof system $\mathcal{P}$, Part 2 of our simulation technique consists of the following step:

5. Show that $\mathcal{P}$ has short eFrege-proofs of the two properties of restriction (Definition 19). That is, show that $\mathcal{P}$ obeys condition-II from above.

5 Cumulator Extraction and NP Oracle simulation

For a complete p-simulation of #SAT proof systems by CLIP+eFrege, we need to apply both part-1 & part-2 of our simulation technique on the same. In the upcoming subsections, we apply Part-1 of our simulation technique to existing #SAT proof systems. Recall that the correctness of our cumulator extraction technique implies that CLIP^{NP} (Definition 8) p-simulates the proof systems which are closed under restrictions (Theorem 32, 40, 47). In the next section, we also apply part-2 of our simulation technique for CPOG, which is sufficient as it can p-simulate all other proof systems.

5.1 Cumulator extraction from KCPS(#SAT)

In this section, we apply our simulation technique to the Knowledge Compilation based Proof System (KCPS(#SAT)). In [12], the authors define a static propositional model counting proof system based on the knowledge representation known as dec-DNNF(see Definition 24). First we redefine the KCPS(#SAT) propositional model counting proof system (Definition 27) from [12]. We then show that KCPS(#SAT) is closed under restrictions (Theorem 28), which is the first step in our simulation technique. Step-3, 4 of the simulation technique are proved in Lemma 30.

Definition 24 (dec-DNNF [25])**.** *A decision Decomposable Negation Normal Form (dec-DNNF) circuit D on variables X is a directed acyclic graph (DAG) with exactly one node of indegree* 0 *called the source. Nodes of outdegree* 0*, called the sinks, are labeled by* 0 *or* 1*. The other nodes have outdegree 2 and can be of two types: decision-nodes or $\wedge$-nodes. Decision nodes are labeled with a variable $x \in X$ and have one outgoing edge labeled with* 1 *and the other labeled by* 0*.*

If there is a decision node in D labeled with variable x, we say that x is tested in D. A valid dec-DNNF has the following properties:

- *Every $x \in X$ is tested at most once on every source-sink path of D.*
- *Every $\wedge$-gate of D is decomposable: there are no common variables tested in both of the dec-DNNFs rooted at the end of its outgoing edges.*

For an example of a dec-DNNF, refer [12, Figure 1]. Given any dec-DNNFD, it is easy to find the number of models of D (denoted by $\#_{\text{models}}(D)$) and it is also easy to test if a total assignment is satisfying D or not. We briefly explain the procedure for the same.

Testing if α is a model of dec-DNNF: Given a dec-DNNFD and a total assignment α, start from the source of D: if the node is a decision node, follow the outgoing edge consistent with α and if the node is an $\wedge$-gate, follow both the outgoing edges. Repeat this process recursively until all the paths followed reach sink-nodes. α is a satisfying assignment of D (i.e., $D(\alpha) = 1$) only if all sink-nodes reached by these paths are 1-sinks, else $D(\alpha) = 0$.

Finding $\#_{\text{models}}(D)$ for a dec-DNNFD: To find the total number of models for a dec-DNNFD, maintain a counter at each node and start from the sinks (in bottom-up fashion): if the node is a sink, assign the value of the sink to the counter. If the node is a decision node, assign its counter with the sum of the children counters. If the node is an $\wedge$-node, assign its counter with the product of the children counters. Finally the counter at the source holds the $\#_{\text{models}}(D)$. This is the general idea, for the detailed algorithm see [11, Proposition 1.57]. As a result, we have the following.

Proposition 25. *[11, Proposition 1.57][12, p.92] Given a dec-DNNF D and a total assignment α to vars(D), there exists polynomial time procedures to find if α is a satisfying assignment of D and also to find the total number of models of D.*

KCPS(#SAT) proof system uses cert-dec-DNNF which is a restriction of dec-DNNF.

Definition 26 (cert-dec-DNNF [12])**.** *A certified dec-DNNF D on variables X is a dec-DNNF such that every 0-sink s of D is labeled with a clause C_s. D is said to be correct if for every assignment α to $vars(X)$ if there is a path from the source of D to a 0-sink (s) following edges according to α, $C_s|_\alpha = \bot$. We denote by $F(D) := \bigwedge_{s \in 0\text{-}sinks(D)} C_s$.*

It is known that Proposition 25 holds even for cert-dec-DNNFs.

Definition 27 (KCPS(#SAT) [12])**.** *Given a CNF Φ, a certificate in the KCPS(#SAT) system that Φ has k satisfying assignments is a correct cert-dec-DNNF D such that*

- *every clause of $F(D)$ is a clause of Φ,*
- *D computes Φ : for every clause $C \in \Phi$, $D \to C$, this can be verified by $(D \wedge \overline{C})$ having 0 satisfying assignments.*
- *D has k satisfying assignments.*

Theorem 28. KCPS(#SAT) *is closed under restrictions.*

Proof. Let Φ be a CNF formula over n variables and cert-dec-DNNF D be the KCPS(#SAT) proof of Φ. Let $|D| = t$. We show that given any partial assignment α to $vars(\Phi)$ there exists a cert-dec-DNNF $D' = D|_\alpha$ of $\Phi|_\alpha$ with size at most t.

Given D and α, we prune the tree starting from the source node: if the node is a decision node of variable x and $x \in vars(\alpha)$, we remove the decision node x (along with its two outgoing edges) and replace it with the child node which was on the consistent outgoing edge of decision-node x w.r.t. α. If the node is an $\wedge$-gate or a decision node of variable $x \notin vars(\alpha)$, we keep both the outgoing edges. Repeat this process recursively until you reach the sinks in all the retained paths. At this point, we have almost finished pruning D only the labels of 0-sinks, in all the retained paths, need to be updated correctly w.r.t. α. To finish pruning, if any of the retained paths in D end in a 0-sink s, update its label C_s with $C_s|_\alpha$. Let us denote this pruned cert-dec-DNNF as D'. For an example of pruning a dec-DNNF, see Example 29. To show that $\#_{\text{models}}(D') = \#_{\text{models}}(\Phi|_\alpha)$, we show below that $D' \leftrightarrow \Phi|_\alpha$.

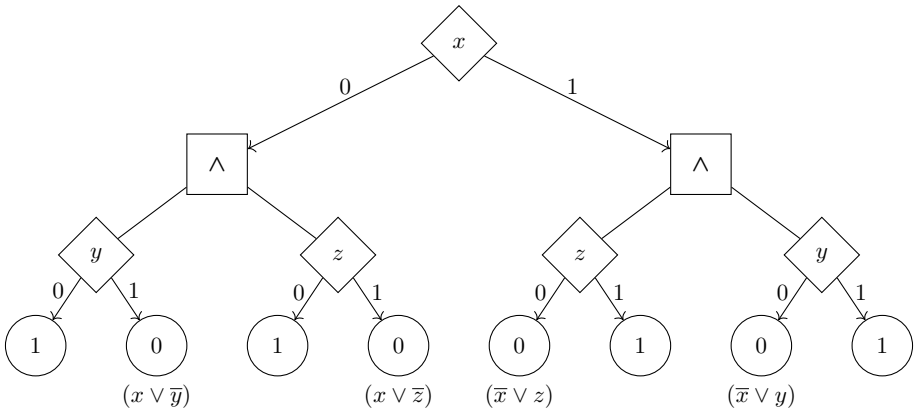
To show $\Phi|_\alpha \to D'$: Since D is correct, we know that $F(D) \subseteq \Phi$. Observe that the above pruning updates the labels of D such that $F(D') \subseteq \Phi|_\alpha$.

For the other direction, we need to show that $D' \to \Phi|_\alpha$: Since D is correct, we know that $D \to C$, for all $C \in \Phi$. That is, for all total assignments γ to $vars(\Phi)$, if $D(\gamma) = 1$ then $C|_\gamma = 1$. We need to prove the following:

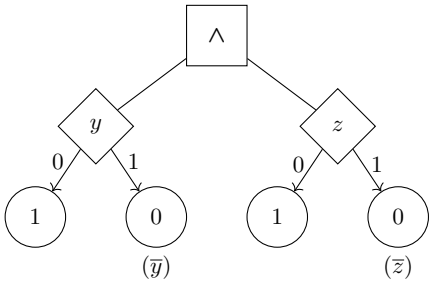
$$D' \to C|_\alpha \text{ for all } C|_\alpha \in \Phi|_\alpha \tag{1}$$

The easy case is when the partial assignment α satisfies all the clauses in Φ. That is, for every clause $C \in \Phi$, α sets at least one of its literals to 1. Then conclusion of equation 1 is always true and we are done.

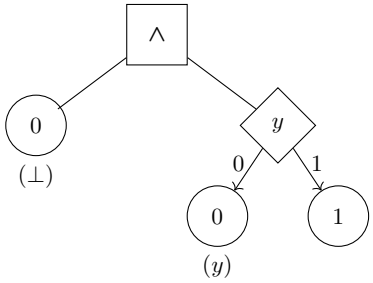
For the remaining partial assignments α, assume for contradiction that equation 1 is not true. Then, there is a total assignment β for $\big(vars(\Phi) \setminus vars(\alpha)\big)$ and a $C|_\alpha \in \Phi|_\alpha$ such that $D'(\beta) = 1$ and $(C|_\alpha)|_\beta = 0$. Note that α and β do not share any common variables and $\alpha \cup \beta$ is a total assignment for $vars(\Phi)$. D' is obtained by pruning D with paths consistent w.r.t. α. We know that β is a model of D', therefore $D'(\beta)$ reaches only 1-sinks. In other words, if one begins with D and starts testing if $\alpha \cup \beta$ is a model, it also ends with all 1-sinks (i.e $D(\alpha \cup \beta) = 1$). But we have $(C_\alpha)|_\beta = C_{\alpha \cup \beta} = 0$ which is a contradiction. This is not possible as $D \to C$ for all $C \in \Phi$ is the assumption. □



(a) cert-dec-DNNF D for CNF Φ [12]



(b) cert-dec-DNNF D' for CNF $\Phi|_{\alpha_1}$ with $\alpha_1 := \{x = 0\}$. Observe that we keep the left-child of decision-node x (since $x = 0$) from D in D'.



(c) cert-dec-DNNF D'' for CNF $\Phi|_{\alpha_2}$ with $\alpha_2 := \{x = 1, z = 0\}$. Observe that we keep the right-child of decision node x (since $x = 1$) and left-child of decision node z (since $z = 0$) from D in D''.

Figure 4: cert-dec-DNNF used in Example 29

Example 29 ([12])**.** *Consider a simple example of a CNF computing $x = y = z$ as $\Phi := (\overline{x} \vee y) \wedge (\overline{y} \vee x) \wedge (\overline{x} \vee z) \wedge (\overline{z} \vee x)$. Φ has 2 models $x = y = z = 0$ and $x = y = z = 1$. A* KCPS(#SAT) *proof of Φ is shown in Figure 4a. The decision nodes are shown as diamond-shaped with the label of its corresponding variable inside it. The $\wedge$-nodes are shown as square-shaped with the label of $\wedge$ inside it. The sink-nodes are shown as circle-shaped with their label of $0/1$ value inside it. All 0-sinks s are additionally labelled with their corresponding C_s clauses just below them.*

Now consider the partial assignment $\alpha_1 := \{x = 0\}$. The CNF $\Phi|_{\alpha_1} := (1) \wedge (\overline{y}) \wedge (1) \wedge (\overline{z})$. Following the pruning procedure of Theorem 28, we build the cert-dec-DNNFD' for $\Phi|_{\alpha_1}$ as shown in Figure 4b. Note that $\#_{\text{models}}(D') = \#_{\text{models}}(\Phi|_{\alpha_1}) = 1$ (i.e $x = y = z = 0$).

Now consider the partial assignment $\alpha_2 := \{x = 1, z = 0\}$. The CNF $\Phi|_{\alpha_2} := (y) \wedge (1) \wedge (\bot) \wedge (1)$. Following the pruning procedure of Theorem 28, we build the cert-dec-DNNF D'' for $\Phi|_{\alpha_2}$ as shown in Figure 4c. Note that $\#_{\text{models}}(D'') = \#_{\text{models}}(\Phi|_{\alpha_2}) = 0$.

Lemma 30. *Given a* KCPS(#SAT) *proof π of Φ and a binary integer $\mathbf{0} \leq J < \mathbf{2}^n$ where $n = |vars(\Phi)|$. There is a polynomial time procedure in $|\pi|$ that returns the cumulative number of models*

of Φ *in* $[\mathbf{0}, J]$ *(up to and including* J *(*$C_{models}(\Phi, J)$*))*

Proof. Find the partial assignment cover PAC(**0**,J) by running Algorithm 1. According to Lemma 11, this PAC(0,J) contains a maximum of n partial assignments. For each of these partial assignments $\alpha \in$ PAC(0,J), we restrict the cert-dec-DNNF D in π to obtain the cert-dec-DNNF D' corresponding to $\Phi|_\alpha$ (as described in Theorem 28). For each such D', we use Proposition 25 to find the individual model-counts of $\Phi|_\alpha$'s. Finally, we add all these counts to return the required $C_{models}(\Phi, J)$. This procedure takes $\mathcal{O}(n.|\pi|)$ time. □

Corollary 31. *There is a polynomial time of extracting a cumulator circuit from a* KCPS(#SAT) *proofs.*

Theorem 32. CLIP^{NP} *simulates* KCPS(#SAT)*.*

Proof. From a KCPS(#SAT) proof we extract the cumulator circuit ξ in polynomial time. Therefore lip(ξ) is polynomial size, and is true because of the correctness we have argued for Theorem 28 and Lemma 11. The NP-oracle finds lip(ξ) in a single line. □

In Krajíček's book on *Bounded Arithmetic, Propositional Logic, and Complexity Theory* [26] it discusses how relationships between bounded arithmetic and propositional proof complexity show that eFrege is capable of the power to simulate any propositional proof system S, provided it is equipped to access the reflection principle of S. The reflection principle codifies the correctness of S in arithmetic, but can be re-translated back into propositional logic through a family of tautologies, that are polynomial-time recognisable. We only use this here for a more concrete statement about simulations without the use of an NP-oracle.

Theorem 33. *There is a family of propositional tautologies* $||\Psi||$ *which can be recognized in polynomial time such that* CLIP $+$eFrege $+||\Psi||$ *simulates* KCPS(#SAT)*.*

Proof. Consider a new proof propositional system *Extract*(KCPS(#SAT)). Recall that a proof system is a function that maps proofs (as strings) to theorems.

$$\textit{Extract}(\mathsf{KCPS}(\#\mathsf{SAT}))(\pi) = \begin{cases} \phi_{\xi,k}, & \pi \text{ is a KCPS(\#SAT) proof of } \#_{\text{models}}(\phi) = k \\ & \text{and } \xi \text{ is the cumulator circuit extracted from it,} \\ \mathsf{eFrege}(\pi), & \text{otherwise.} \end{cases}$$

Extract(KCPS(#SAT)) is complete since it accepts any eFrege proof. Its soundness is shown through correctness of the cumulator extraction of Lemma 30. Hence we can add *Extract*(KCPS(#SAT)) to CLIP. What we end up getting is that CLIP $+$*Extract*(KCPS(#SAT)) almost trivially simulates KCPS(#SAT). The only thing we have to provide is the cumulator circuit which is extracted from KCPS(#SAT) via Cor. 31. The rest is trivial because every KCPS(#SAT) proof is automatically accepted by *Extract*(KCPS(#SAT)), proving exactly the propositional statement we want. There is no conversion of proof needed.

Every propositional proof system can be simulated by eFrege plus a polynomial-time recognisable set of tautologies [26]. *Extract*(KCPS(#SAT)) is simulated by eFrege $+$ $\|\text{refl}(\textit{Extract}(\mathsf{KCPS}(\#\mathsf{SAT})))\|$, where $\|\text{refl}(S)\|$ is a p-time recognisable set of propositions that encode an arithmetic statement of correctness of S, for propositional proof system S. □

5.2 Cumulator extraction from MICE′

In this section, we apply our simulation technique to MICE′ (Model-counting Induction by Claim Extension) [21, 5]. MICE′ is a propositional model counting proof system which is shown to be p-equivalent to MICE [21] but has simpler inference-rules. Hence extracting cumulator circuits from MICE′ proofs is sufficient. First we redefine the MICE′ (Definition 34) from [5]. We then show that MICE′ is closed under restrictions (Theorem 36). Step-3, 4 of the simulation technique are proved in Lemma 38.

Definition 34 (MICE′ [5])**.** *Let Φ be a CNF with k models. A* MICE′ *derivation of $L_f := (\Phi, \emptyset, k)$ from Φ is a sequence $\pi = L_1, ..., L_f$ of claims where each $L_i = (F_i, A_i, c_i)$ keeps track of the number of models of the CNF $(F_i|_{A_i})$ as c_i. These claims are derived using the rules in Figure 5.* MICE′ *has two complexity measures, steps(π) counts the number of inference steps whereas size(π) includes the size of the Resolution refutations ($|\rho|$) used in the Composition-lines of the proof as well.*

Axiom:

$$\frac{}{(\emptyset, \emptyset, \mathbf{1})}$$

Composition:

$$\frac{(F, A_1, c_1) \dots (F, A_n, c_n)}{(F, A, \Sigma_{i\in[n]} c_i),\ \rho}$$

(C1) $vars(A_1) = \cdots = vars(A_n)$ and $A_i \neq A_j$ for $i \neq j$
(C2) $A \subseteq A_i$ for all $i \in [n]$
(C3) $\rho := (A \cup F \cup \{\overline{A}_i \mid i \in [n]\}) \vdash_{Res} \bot$

Join:

$$\frac{(F_1, A_1, c_1) \qquad (F_2, A_2, c_2)}{(F_1 \cup F_2, A_1 \cup A_2, c_1 \cdot c_2)}$$

(J1) $A_1 \simeq A_2$
(J2) $var(F_1) \cap var(F_2) \subseteq A_i \mid i \in [n]$

Extension:

$$\frac{(F_1, A_1, c_1)}{(F, A, c_1 \cdot \mathbf{2}^{|vars(F)\setminus(vars(F_1)\cup vars(A))|})}$$

(E1) $F_1 \subseteq F$
(E2) $A|_{vars(F_1)} = A_1$
(E3) A satisfies $F \setminus F_1$

Figure 5: MICE′ derivation rules

In the proof of Theorem 36, we need the following definition.

Definition 35. *[Invalid* MICE′ *claims] Let Φ be a CNF formula and α be a partial assignment to vars(Φ). Let π be a* MICE′ *proof of Φ and $L = (F, A, c)$ be any claim in π. We say that any claim $L' = (F|_\alpha, A \setminus vars(\alpha), c')$ is invalid w.r.t α if any of the following holds:*

a. the formula $F|_\alpha$ has an empty clause in it.

b. all the hypothesis claims of L in π are invalid w.r.t α and $A \not\simeq \alpha$.

c. if at least one hypothesis claim of L in π is invalid w.r.t. α and L was derived by the Join-rule in π.

d. if L was derived by an Extension-rule in π and L' does not satisfy Condition-E3 of the Extension-rule.

Observe that in cases b,c and d above, the models in c were dependent on the restriction of A which is inconsistent with α.

Theorem 36. MICE′ *is closed under restrictions.*

Proof. Let Φ be a CNF formula and $\pi := \{L_1, ..., L_f\}$ be a MICE′ proof of Φ of size s. We show that given any partial assignment α there exists a MICE′ proof $\pi' : \{L'_1, ..., L'_f\}$ of $\Phi|_\alpha$ of size at most s.

We first deal with the partial assignments which satisfy or falsify the CNF formula Φ: Firstly, if α is a partial/ total assignment which satisfies Φ (i.e α satisfies every clause $\in \Phi$), π' is derived as follows: $(\emptyset,\ \emptyset,\ \mathbf{1})$ by the axiom-rule, $(\Phi|_\alpha, \emptyset, \mathbf{2}^{|vars(\Phi|_\alpha)|})$ by extension-rule. Secondly, if α is a partial/ total assignment which falsifies Φ (i.e α falsifies every literal in a clause $\in \Phi$), π' is derived as follows: $\frac{}{(\Phi|_\alpha,\ \emptyset,\ \mathbf{0}),\ \rho}$ by using the composition-rule with 0 hypothesis where ρ is the Resolution refutation of $\Phi|_\alpha$.

Now we are left with partial assignments α for which $\Phi|_\alpha$ is a nontrivial CNF. In this case, we build π' from π in two passes. In the first pass using induction on the size of π, we derive π'-claims and also mark some of them as invalid (Ref. definition 35) in the process. In the second pass, we discard all the invalid claims and the resultant is the required π'.

Phase I:

Induction Statement: Let Φ be a CNF formula and α be a partial assignment such that $\Phi|_\alpha$ is a non-trivial CNF. Given a MICE′ proof $\pi = \{L_1, .., L_f\}$ of Φ, by induction on $i \in [f]$ we show that corresponding to $L_i := (F_i, A_i, c_i) \in \pi$ we can derive the claim $L'_i := (F_i|_\alpha, A_i \setminus vars(\alpha), c'_i) \in \pi'$ such that if L'_i is not invalid (as discussed above), then $c'_i = \#_{\text{models}}(F_i|_{\alpha \cup A_i \setminus vars(\alpha)})$.

Base case: For the base case $i = 1$, L_i could either be derived by an axiom-rule or a composition-rule with 0 hypothesis. In the former case, L'_i is equal to L_i and it is valid. In the latter case i.e $\frac{}{(F_i,\emptyset,\mathbf{0}),\rho}$, $\rho := F_i \vdash_{Res} \bot$ and as Resolution is closed under restrictions, $\rho|_\alpha := F_i|_\alpha \vdash_{Res} \bot$. Therefore $L'_i = (F_i|_\alpha, \emptyset, 0), \rho|_\alpha$ and it is valid. Both the cases satisfy the induction statement as the empty CNF is always true and unsatisfiable CNF is always false.

Inductive Step: For the inductive step assume that the statement is true until $i-1$. $L_i \in \pi$ must be derived from one of the four MICE′ rules. For each of the rules, we show below that the induction statement holds for L'_i as well:

1. If L_i is derived by an axiom-rule, L'_i is also the same claim.
2. If L_i is derived by an Extension-rule as follows:

$$\frac{L_j := (F_1,\ A_1,\ c_1)}{L_i := (F,\ A,\ c_1.\mathbf{2}^{|vars(F)\setminus(vars(F_1)\cup vars(A))|})}, j < i$$

   for the corresponding claim L'_i in π', following four cases can occur.

   (a) The corresponding hypothesis claim $L'_j := (F_1|_\alpha,\ A_1 \setminus vars(\alpha),\ c'_1)$ in π' is valid and the restriction $A \simeq \alpha$. Then set

$$L'_i := (F|_\alpha,\ A \setminus vars(\alpha),\ c'_1 \cdot \mathbf{2}^{|vars(F|_\alpha)\setminus(vars(F_1|_\alpha)\cup vars(A\setminus vars(\alpha))|}).$$

   This is sound as the Conditions E1, E2, E3 are satisfied in this case.

   (b) The corresponding hypothesis claim $L'_j := (F_1|_\alpha,\ A_1 \setminus vars(\alpha),\ c'_1)$ in π' is valid but the restriction $A \not\simeq \alpha$.

   - Verify if $\{A \setminus vars(\alpha)\}$ satisfies $(F|_\alpha \setminus F_1|_\alpha)$. If yes, set

$$L'_i := (F|_\alpha,\ A \setminus vars(\alpha),\ c'_1.\mathbf{2}^{|vars(F|_\alpha)\setminus(vars(F_1|_\alpha)\cup vars(A\setminus vars(\alpha))|}).$$

     This is sound as the Conditions E1, E2 are satisfied and we specifically verified that E3 is also satisfied.

- If no, mark L'_i in π' as an invalid claim.

(c) The corresponding hypothesis claim $L'_j := (F_1|_\alpha,\ A_1 \setminus vars(\alpha),\ c'_1)$ in π' is either invalid or valid but CNF $F|_\alpha$ has a empty-clause in it. Mark L'_i in π' as an invalid claim in this case.

3. If L_i is derived by a Join-rule as follows:

$$\frac{L_j := (F_1,\ A_1,\ c_1), L_k := (F_2,\ A_2,\ c_2)}{L_i := (F_1 \cup F_2,\ A_1 \cup A_2,\ c_1.c_2)}, \quad j,k < i$$

for the corresponding claim L'_i in π', following two cases can occur.

(a) The corresponding hypothesis claims $L'_j := (F_1|_\alpha,\ A_1 \setminus vars(\alpha),\ c'_1), L'_k := (F_2|_\alpha,\ A_2 \setminus vars(\alpha),\ c'_2)$ in π' are valid. Then set

$$L'_i := ((F_1 \cup F_2)|_\alpha,\ A_1 \setminus vars(\alpha) \cup A_2 \setminus vars(\alpha),\ c'_1.c'_2)$$

This is sound as the Conditions J1, J2 are satisfied in this case.

(b) One or both of the corresponding hypothesis claims L'_j, L'_k are invalid. In this case, mark L'_i in π' as an invalid claim.

4. If L_i is derived by a Composition-rule as follows:

$$\frac{L_{k_1} := (F,\ A_1,\ c_1)...L_{k_n} := (F,\ A_n,\ c_n)}{L_i := (F,\ A,\ \Sigma_{i\in[n]}c_i),\ \rho}, \quad k_1,...,k_n < i$$

for the corresponding claim in π', following four cases can occur.

(a) The corresponding hypothesis claims $L'_{k_1} := (F|_\alpha,\ A_1 \setminus vars(\alpha),\ c'_1), ..., L'_{k_n} := (F|_\alpha,\ A_n \setminus vars(\alpha),\ c'_n)$ in π' are valid. Then set

$$L'_i := (F|_\alpha,\ A \setminus vars(\alpha),\ \Sigma_{i\in[n]}c'_i),\ \rho|_\alpha.$$

This is sound as the Conditions C1, C2 are satisfied in this case and C3 is satisfied as Resolution is closed under restrictions.

(b) One or more of the hypothesis claims (say $L'_{k_1} := (F|_\alpha,\ A_1 \setminus vars(\alpha),\ c'_1), ..., L'_{k_j} := (F|_\alpha,\ A_j \setminus vars(\alpha),\ c'_j)$) are invalid but $A \simeq \alpha$. Then set

$$L'_i := (F|_\alpha,\ A \setminus vars(\alpha),\ \Sigma_{i\in[j+1,n]}c_i),\ \rho|_\alpha$$

This is sound as C1, C2 are obviously satisfied and C3 is satisfied as follows: Since $F|_\alpha$ is same among all hypothesis, the invalid hypothesis should be a result of $A_1, ..., A_j \not\simeq \alpha$. π being a valid MICE′ proof, we know that $\rho := (F \cup A \cup \{\overline{A_i} | i \in [n]\}) \vdash_{\overline{Res}} \perp$ now because $A_1, ..., A_j \not\simeq \alpha$ restricting $\rho|_\alpha := (F|_\alpha \cup A \setminus vars(\alpha) \cup \{\overline{A_i} | i \in [j+1, n]\}) \vdash_{\overline{Res}} \perp$ as imposing α satisfies the clauses representing $\overline{A_1}, ..., \overline{A_j}$ and hence can be dropped.

(c) All the hypothesis claims are invalid and $A \not\simeq \alpha$ or $F|_\alpha$ has a empty-clause in it. In this case, mark the L'_i in π' as invalid.

Note that the last claim L'_f was definitely not marked as invalid in the above process. This is because the last inference would be only from either a Composition-rule or a Join-rule with hypothesis restrictions $A_1, A_2 = \emptyset$. In the former case, L_f is the claim $(\Phi, \emptyset, k)$ and $\emptyset \simeq \alpha$ so this is the case 4a or 4b above, both of these do not result in invalid claims. In the latter case with join-rule, L_f would be derived as follows: $\frac{(F_1,\ \emptyset,\ c_1),(F_2,\ \emptyset,\ c_2)}{L_f:=(\Phi:=F_1\cup F_2,\ \emptyset,\ c_1.c_2)}$. In the case 3a above, L'_f is definitely valid and we can see that this is the only possibility as case 3b would be only possible when $\Phi|_\alpha$ has a empty-clause in it and this case was handled before the start of phase-I itself.

Phase-II : Finally, prune through the π' from the last line L'_f (which as discussed above is valid) and recursively include the valid hypothesis used at every line until you reach the axiom clause. This will get rid of all the invalid claims and the redundant claims which are deriving these invalid claims. Now π' is a correct MICE′ proof of $\Phi|_\alpha$ as every retained inference was shown to be sound and satisfying all the conditions of MICE′ inference rules. □

Example 37. *Consider a simple example of a CNF computing $x = y = z$ as $\Phi := (\overline{x} \vee y) \wedge (\overline{y} \vee x) \wedge (\overline{x} \vee z) \wedge (\overline{z} \vee x)$. Φ has 2 models $x = y = z = 0$ and $x = y = z = 1$. A MICE′ proof of the same is as follows:*

$\pi := \Big\{$ $(\emptyset,\ \emptyset,\ 1)$ *by axiom,*
$(\Phi,\ x = y = z = 0,\ 1)$ *by Extension,*
$(\Phi,\ x = y = z = 1,\ 1)$ *by Extension,*
$(\Phi,\ \emptyset,\ 2)$ *by Composition with* $\rho := (x \vee z), (\overline{x} \vee \overline{z}), (z), (\overline{z}), \bot \Big\}$.

Now consider a partial assignment $\alpha := \{x = 0\}$. Following the above procedure, we build π' for $\Phi|_\alpha$ as follows.

$\pi' := \Big\{$ $(\emptyset,\ \emptyset,\ 1)$ *by axiom,*
$(\Phi|_\alpha,\ y = z = 0,\ 1)$ *by Extension,*
<Invalid-claim>,
$(\Phi|_\alpha,\ \emptyset,\ 1)$ *by Composition with* $\rho|_\alpha := (z), \bot \Big\}$

We derive π' using the procedure defined in proof of Theorem 36. Precisely, claim 1 of π' is derived from case 1 and claim 2 is derived from case 2b. Claim 3 is declared as invalid from case 2b as $y = z = 1$ does not satisfy $\Phi|_\alpha := (\overline{y}) \wedge (\overline{z})$). Finally, claim 4 is derived from case 4b.
Now starting from the last claim of π', we retain it and include its valid hypothesis claims. So the final $\pi' := \{(\emptyset,\ \emptyset,\ 1), (\Phi|_\alpha,\ y = z = 0,\ 1), (\Phi|_\alpha,\ \emptyset,\ 1), \rho|_\alpha := (z), \bot\}$. This is valid MICE′ proof of $\Phi|_\alpha$ with the correct model-count.

Lemma 38. *Given a MICE′ proof π of Φ and a binary integer $0 \leq J < 2^n$ where $n = |vars(\Phi)|$. There is a polynomial time procedure in $|\pi|$ that returns the number of models of Φ in $[0, J]$ i.e $(C_{models}(\Phi, J))$).*

Proof. Run Algorithm 1 on (J, n) to find the disjoint binary partial assignment cover PAC(0,J). According to Lemma 11, this PAC(0,J) contains a maximum of n partial assignments. For each of these partial assignments $\alpha \in$ PAC(0,J), follow the procedure described in Theorem 36 (which takes $|\pi|$ time) to find the MICE′ proof $\pi' = L'_1, ..., L'_f$ of $\Phi|_\alpha$. Add the number of models (c') in the last line ($L'_f = (\Phi|_\alpha, \emptyset, c'_f)$) of all the above restricted MICE′ proofs (π's) to get $C_{models}(\Phi, J)$. This procedure takes $\mathcal{O}(n.|\pi|)$ time. □

Corollary 39. *From a MICE′ proof π of Φ you can efficiently extract cumulator circuits.*

Theorem 40. CLIP^{NP} *simulates* MICE′.

Proof. Corollary 39 demonstrates that one can extract cumulator circuits of polynomial size from any MICE′ proof. The proposition $\mathsf{lip}(\xi)$ is polynomial in the original propositional formula. Since these circuits are correct, owing to Theorem 36 and Lemma 11, the NP-Oracle in CLIP^{NP} always returns true. □

Theorem 41. *There is a family of propositional tautologies $||\Psi||$ which can be recognized in polynomial time such that* CLIP $+$eFrege $+||\Psi||$ *simulates* MICE′.

Proof. This works the same as the proof of Theorem 33 but we substitute MICE′ for KCPS(#SAT). □

We will later show (Corollary 54) that $||\Psi||$ is empty in both Theorem 32 and 41.

5.3 Cumulator extraction from CPOG

In this section, we apply our simulation technique Part-1 to the CPOG (Certified Partitioned-Operation Graphs) [7] proof system. That is, we show that CPOG is closed under restrictions (Lemma 45). For a CNF formula ϕ, CPOG is a propositional weighted-model counting proof system which consists of a POG structure G (Definition 42) along with Resolution proofs of $\phi \leftrightarrow G$. It is well known that model counting is easy for POG [7, p. 16]. However, given a POG G and a CNF ϕ, verifying that $G \equiv \phi$ is hard. In this paper, we only study CPOG for the unweighted (standard) model counting. We first define the POG structure (Definition 42) and the CPOG proof system (Definition 44) which is based on POG from [7].

Definition 42 (POG [7])**.** *A Partitioned-Operation Graph (POG) (say G) is a directed acyclic graph defined on n variables (say X). Each node v in a POG has an associated dependency set $\mathcal{D}(v) \subseteq X$ and a set of models $\mathcal{M}(v)$, consisting of all complete assignments that satisfy the formula represented by the POG rooted at v. The leaf nodes (with outdegree $= 0$) can be of the following:*

- *Boolean constants* 0 *or* 1*. Here, $\mathcal{D}(1) = \mathcal{D}(0) = 0$, $\mathcal{M}(0) = \emptyset$ and $\mathcal{M}(0) = \langle X \rangle$.*
- *Literal l for some variable x such that $vars(l) = x \in X$. Here, $\mathcal{D}(l) = x$, $\mathcal{M}(l) = \{\alpha \in \langle X \rangle \mid \alpha(x) \equiv l\}$.*

The rest of the nodes (internal nodes) can be of the following:

- *Decomposable AND-gate ($\wedge^p$) [1] with outgoing edges to $v_1, \ldots, v_k$ for $k > 1$. Here, $\mathcal{D}(\wedge^p) = \bigcup_{1 \le i \le k} \mathcal{D}(v_i)$ and $\mathcal{M}(\wedge^p) = \bigcap_{1 \le i \le k} \mathcal{M}(v_i)$. This node needs to follow the decomposable property namely, $\mathcal{D}(v_i) \cap \mathcal{D}(v_j) = \emptyset$ for every $i, j \in [k]$ and $i \neq j$.*
- *Deterministic OR-gate ($\vee^p$) with outgoing edges to v_1, v_2. Here, $\mathcal{D}(\vee^p) = \mathcal{D}(v_1) \cup \mathcal{D}(v_2)$ and $\mathcal{M}(\vee^p) = \mathcal{M}(v_1) \cup \mathcal{M}(v_2)$. This node needs to follow the deterministic property namely, $\mathcal{M}(v_1) \cap \mathcal{M}(v_2) = \emptyset$.*

The edges of G have an optional polarity to indicate if they need to be negated (polarity $= 1$) or not (polarity $= 0$). Here, $\mathcal{D}(\neg v) = \mathcal{D}(v)$ and $\mathcal{M}(\neg v) = \langle X \rangle - \mathcal{M}(v)$. Every POG has a designated root node r with indegree $= 0$.

[1] For simplicity, we use the same notations from [7]. Here, p stands for partitioned-operation formulas.

For an example of POG see [8].

The weighted model counting can be seen as a ring-evaluation problem for a commutative ring over rational numbers $\in [\mathbf{0}, \mathbf{1}]$. The ring evaluation problem takes a weight function $w(x) \in [\mathbf{0}, \mathbf{1}]$ for all variables $x \in X$ and computes the following:

$$R(v, w) = \Sigma_{\alpha \in \mathcal{M}(v)} \, \Pi_{l \in \alpha} \, w(l) \tag{2}$$

where $w(\overline{x}) = \mathbf{1} - w(x)$. For standard unweighted model-counting (i.e $|\mathcal{M}(v)|$), one can fix $w(x) = \frac{\mathbf{1}}{\mathbf{2}}$ for all $x \in X$ and $|\mathcal{M}(v)| = \mathbf{2}^{|\mathbf{X}|} \cdot R(v, w)$. The following properties of the ring evaluation function are well known.

Proposition 43 ([7])**.** *Ring evaluations for operations $\neg$, $\wedge^p$ and $\vee^p$ satisfies the following for any weight function w: (i) $R(\neg v, w) = \mathbf{1} - R(v, w)$, (ii) $R(\bigwedge_{1 \leq i \leq k}^{p} v_i, w) = \prod_{1 \leq i \leq k} R(v_i, w)$, (iii) $R(v_1 \vee^p v_2, w) = R(v_1, w) + R(v_2, w)$.*

For a CNF ϕ, a CPOG proof π consists of a POG G such that $G \equiv \phi$. However, to make the proof easily verifiable, π explicitly has the proof that G is a POG and is equivalent to ϕ. We present the precise definition following [3] below.

Definition 44 (CPOG [3, 7])**.** *A CPOG proof π of ϕ is the tuple $(\mathcal{E}(G), \delta, \rho, \psi)$, where*

- *G is a POG such that $G \equiv \phi$ and $\mathcal{E}(G)$ is a clausal encoding of the POG G by defining an extension variable for every internal node of G (That is, if there is an $\wedge^p$ node v with outgoing edges to v_1, v_2 and a negated edge to v_3, we add the extension variable $e_v \leftrightarrow (v_1 \wedge v_2 \wedge \overline{v_3})$).*
- *δ is the determinism proof for OR-gates which contains a Resolution proof of $\mathcal{E}(G) \wedge (v_1) \wedge (v_2)$ for every $\vee^p$-gate v with outgoing edges to v_1, v_2.*
- *ρ is the forward implication proof (i.e. $\phi \models G$) consisting of a Resolution proof of $\mathcal{E}(G) \wedge \phi \wedge (\overline{r})$.*
- *ψ is the reverse implication proof (i.e. $G \models \phi$) consisting of a Resolution proof of $\mathcal{E}(G) \wedge (r) \wedge \overline{C}$ for every clause $C \in \phi$.*

Observe that the extension variables used in CPOG are restrictive as compared to those in eFrege.

Lemma 45. CPOG *is closed under restrictions.*

Proof. For a given CNF ϕ, a CPOG proof consists of a POG G and a Resolution proof of $\phi \leftrightarrow G$. A POG is closed under conditioning as for any partial assignment α: replace the inputs labelled by x with $\alpha(x)$ for every x assigned by α and the resulting structure is still a POG G'. This is because, constants are allowed in POG and the $\wedge^p$-nodes will remain decomposable since we are only reducing variables. Also, the $\vee^p$-gates will remain deterministic because if A and B has disjoint models, so are $A|_\alpha$ and $B|_\alpha$. The Resolution proof witness is closed under restrictions. The CPOG proof of $\phi \leftrightarrow G$ uses Resolution proofs which are closed under restrictions, it implies $\phi|_\alpha \leftrightarrow G|_\alpha$. □

This completes Step 1 of our simulation technique for CPOG. Thus we have the following:

Corollary 46. *There is a polynomial time method of extracting a cumulator circuit from a* CPOG *proof.*

Proof. With a CPOG proof, given an assignment α, in polynomial time we can calculate the PAC(0, α) via Fenwick's method (Algorithm 1) and use closure under restrictions to find the values for the sum. By formalising these steps into a circuit as in Definition 17 we get the cumulator circuit. □

Theorem 47. CLIP^{NP} *simulates* CPOG.

Proof. Corollary 46 demonstrates that one can extract cumulator circuits of polynomial size from any CPOG proof. The proposition lip(ξ) is polynomial in the original propositional formula. Since these circuits are correct, owing to Theorem 45 and Lemma 11, the NP-Oracle in CLIP^{NP} always returns true. □

6 CLIP+eFrege p-simulates CPOG

In this section, we apply part-2 of our simulation technique for CPOG. That is, we prove that CPOG admits easy eFrege proofs for the properties of restriction (Lemma 52) which by Theorem 53 leads to CLIP+eFrege p-simulating CPOG. This is sufficient to show that CLIP+eFrege p-simulates all existing #SAT proof systems (Theorem 53, Corollary 54), as in [3], the authors show that CPOG in turn p-simulates MICE′ and KCPS(#SAT). The proofs in this section need to prove basic properties of arithmetic in short eFrege proofs. We can consider the handling of basic arithmetic gates as academic folklore but we provide some sketch overviews in Appendix B to reassure the reader.

Recall, the weight function w is defined for all the variables (X) of POG G as $\frac{\mathbf{1}}{\mathbf{2}}$. In this case, the value of $R(r, w) = \mathbf{2}^{|\mathbf{X}|} \cdot |\mathcal{M}(r)|$. For an assignment α, conditioning the POG G with α (Lemma 45) will give POG G' with root r' and the following will hold $R(r', w) = \mathbf{2}^{|\mathbf{X}-|\alpha||} \cdot |\mathcal{M}(r)|_\alpha|$.

Instead of changing the POG structure, we change the weight function as defined below to obtain the same model count as above i.e. $R(r, w_\alpha) = \mathbf{2}^{|\mathbf{X}-|\alpha||} \cdot |\mathcal{M}(r)|_\alpha|$.

Definition 48. *Given a CNF ϕ and an initial assignment α defined on $x_{n-1} \dots x_i$ and undefined on $x_{i-1} \dots x_0$, we define the weight w_α which weighs variables according to the following*

$$w_\alpha(x_j) = \begin{cases} \mathbf{1} & j \geq i \ \&\ \alpha(x_j) = 1, \\ \mathbf{0} & j \geq i \ \&\ \alpha(x_j) = 0, \\ \frac{\mathbf{1}}{\mathbf{2}} & j < i. \end{cases}$$

Next, in Lemma 49,50,51, we prove some properties of $R(v, w_\alpha)$ for every node v of the POG recursively. We use the general properties of the ring function from Proposition 43 in these proofs. Informally, in Lemma 49, we show that if α was undefined on x and x is not in the dependency set of v (i.e. $x \notin \mathcal{D}(v)$), the value of R does not change when weight function is changed to w_{α_0} or w_{α_1} where $\alpha_b = \alpha \cup \{x = b\}$. In Lemma 50, we show that if x is in $\mathcal{D}(v)$, the values of R hold a weaker relation of $R(v, w_\alpha) = \frac{\mathbf{1}}{\mathbf{2}}(R(v, w_{\alpha_0}) + R(v, w_{\alpha_1}))$. We consider complete assignments α in Lemma 51 and prove that the function $R(v, w_\alpha)$ returns $\mathbf{1}$ if α is a satisfying assignment of the POG rooted at v and $\mathbf{0}$ otherwise. We use these Lemmas to prove that CPOG has easy eFrege proofs of the properties of restriction in Lemma 52.

Lemma 49. *Let α be an initial partial assignment defined up to x_i where $i > 0$. We can prove in the structure of the POG that $R(v, w_\alpha) = R(v, w_{\alpha_0}) = R(v, w_{\alpha_1})$ when x_{i-1} is not in the dependency set of v. This proof can be formalised in a short* eFrege *proof.*

Proof. In all cases, except at leaves, the dependency set is the union of the dependency sets of its children.

Boolean leaf: $R(1, w_\alpha) = \mathbf{1}$ and $R(0, w_\alpha) = \mathbf{0}$ independent of α. Therefore the Lemma statement is easily derived in this case.
Variable leaf: Let the variable leaf be x_j. $R(x_j, w)$ takes the value of $w(x_j)$. If $x_{i-1} \notin \mathcal{D}(x_j)$, then either $j \geq i$ or $j < i - 1$. This is formalised in a tautological disjunction in eFrege.

In either case we show equality is easily derived. If $j \geq i$ then $w_\alpha(x_j) = \mathbf{1}$ when $\alpha(j) = 1$ which also extends to $\alpha_0(j) = 1$ and $\alpha_1(j) = 1$ in which case $w_{\alpha_0}(x_j) = \mathbf{1}$ and $w_{\alpha_1}(x_j) = \mathbf{1}$. Similarly all $w_\alpha(x_j) = w_{\alpha_0}(x_j) = w_{\alpha_1}(x_j) = \mathbf{0}$ when $\alpha(j) = 0$. If $j < i - 1$ then $\alpha, \alpha_0, \alpha_1$ are all undefined on x_j, so the weights are all $\frac{1}{2}$.
Negation: Using the induction hypothesis on the child node c, $R(c, w_\alpha) = R(c, w_{\alpha_0}) = R(c, w_{\alpha_1})$ and so $R(\neg c, w_\alpha) = \mathbf{1} - R(c, w_\alpha) = \mathbf{1} - R(c, w_{\alpha_0}) = \mathbf{1} - R(c, w_{\alpha_1})$ therefore $R(\neg c, w_\alpha) = R(\neg c, w_{\alpha_0}) = R(\neg c, w_{\alpha_1})$.
Partition Conjunction: Let C be the set of child nodes for $\wedge^p$. Using the induction hypothesis $R(c, w_\alpha) = R(c, w_{\alpha_0}) = R(c, w_{\alpha_1})$ for each child $c \in C$. We get the following equality for the products $\Pi_{c \in C} R(c, w_\alpha) = \Pi_{c \in C} R(c, w_{\alpha_0}) = \Pi_{c \in C} R(c, w_{\alpha_1})$. Thus $R(\bigwedge^p_{c \in C}, w_\alpha) = R(\bigwedge^p_{c \in C}, w_{\alpha_0}) = R(\bigwedge^p_{c \in C}, w_{\alpha_1})$.
Partition Disjunction: Let the child nodes of $\vee^p$ be c, d. Using the induction hypothesis $R(c, w_\alpha) = R(c, w_{\alpha_0}) = R(c, w_{\alpha_1})$ and $R(d, w_\alpha) = R(d, w_{\alpha_0}) = R(d, w_{\alpha_1})$.
$R(c \vee^p d, w_\alpha) = R(c, w_\alpha) + R(d, w_\alpha) = R(c, w_{\alpha_0}) + R(d, w_{\alpha_0}) = R(c \vee^p d, w_{\alpha_0})$. Similarly, it can be derived that $R(c \vee^p d, w_\alpha) = R(c \vee^p d, w_{\alpha_1})$.

Each inductive step involves a bounded application of implications using the definitions hence we get short eFrege proofs. □

Lemma 50. *Let α be an initial partial assignment defined up to x_i where $i > 0$. We can prove in the structure of the POG that $R(v, w_\alpha) = \frac{\mathbf{1}}{\mathbf{2}} \cdot (R(v, w_{\alpha_0}) + R(v, w_{\alpha_1}))$. Furthermore we can formalise this in short* eFrege *proofs.*

Proof. If $x_{i-1} \notin \mathcal{D}(v)$, this directly holds from Lemma 49, along with arithmetic properties i.e. $a = \frac{\mathbf{1}}{\mathbf{2}} \cdot (a + a)$. So here we only consider the case that $x_{i-1} \in \mathcal{D}(v)$.
Variable leaf: Let the variable leaf be x_j. $x_{i-1} \in \mathcal{D}(v)$ implies that $j = i - 1$, then $R(v, w_\alpha) = \frac{\mathbf{1}}{\mathbf{2}}$. $R(v, w_{\alpha_0}) = \mathbf{1} \to R(v, w_{\alpha_1}) = \mathbf{0}$ and $R(v, w_{\alpha_0}) = \mathbf{0} \to R(v, w_{\alpha_1}) = 1$, in both cases they sum to 1 which is the right identity when multiplied with $\frac{1}{2}$.
Negation: Using the induction hypothesis on the child node c, i.e. $R(c, w_\alpha) = \frac{1}{2} \cdot (R(c, w_{\alpha_0}) + R(c, w_{\alpha_1}))$, it implies the following.

$$\begin{aligned} R(\neg c, w_\alpha) &= \mathbf{1} - R(c, w_\alpha) = \mathbf{1} - \tfrac{\mathbf{1}}{\mathbf{2}} \cdot (R(c, w_{\alpha_0}) + R(c, w_{\alpha_1})) \\ &= \mathbf{1} - \tfrac{\mathbf{1}}{\mathbf{2}} \cdot (\mathbf{1} - R(\neg c, w_{\alpha_0}) + \mathbf{1} - R(\neg c, w_{\alpha_1})) \\ &= \mathbf{1} - \tfrac{\mathbf{1}}{\mathbf{2}} \cdot (\mathbf{2} - R(\neg c, w_{\alpha_0}) - R(\neg c, w_{\alpha_1})) \\ &= \tfrac{\mathbf{1}}{\mathbf{2}} \cdot (R(\neg c, w_{\alpha_0}) + R(\neg c, w_{\alpha_1})). \end{aligned}$$

Partition Conjunction: Let C be the set of child nodes for $\wedge^p$. Observe that, $x_{i-1} \in \mathcal{D}(c^*)$ for exactly one child c^*. Along with multiplicative commutativity and associativity, we know the following from Proposition 43:
$R(\bigwedge^p_{c \in C} c, w_\alpha) = R(c^*, w_\alpha) \cdot \prod_{c \in \{C \setminus c^*\}} R(c, w_\alpha)$.
Using the induction hypothesis on c^* we get

$$= \tfrac{\mathbf{1}}{\mathbf{2}} \cdot (R(c^*, w_{\alpha_0}) + R(c^*, w_{\alpha_1})) \cdot \prod_{c \in \{C \setminus c^*\}} R(c, w_\alpha).$$

We can use left-distributivity to get

$$= \tfrac{\mathbf{1}}{\mathbf{2}} \cdot R(c^*, w_{\alpha_0}) \cdot \prod_{c \in \{C \setminus c^*\}} R(c, w_\alpha) + \tfrac{\mathbf{1}}{\mathbf{2}} \cdot R(c^*, w_{\alpha_1}) \cdot \prod_{c \in \{C \setminus c^*\}} R(c, w_\alpha).$$

At this point we know that $x_{i-1} \notin \mathcal{D}(c \in \{C \setminus c^*\})$, using Lemma 49 we get

$$= \tfrac{\mathbf{1}}{\mathbf{2}} \cdot R(c^*, w_{\alpha_0}) \cdot \prod_{c \in \{C \setminus c^*\}} R(c, w_{\alpha_0}) + \tfrac{\mathbf{1}}{\mathbf{2}} \cdot R(c^*, w_{\alpha_1}) \cdot \prod_{c \in \{C \setminus c^*\}} R(c, w_{\alpha_1})$$

$$= \tfrac{\mathbf{1}}{\mathbf{2}} \cdot R(\bigwedge^p_{c\in C} c, w_{\alpha_0}) + \tfrac{\mathbf{1}}{\mathbf{2}} \cdot R(\bigwedge^p_{c\in C} c, w_{\alpha_1}) = \tfrac{\mathbf{1}}{\mathbf{2}} \cdot (R(\bigwedge^p_{c\in C} c, w_{\alpha_0}) + R(\bigwedge^p_{c\in C} c, w_{\alpha_1})).$$

Partition Disjunction: Let the child nodes of $\vee^p$ be c, d.

$R(c \vee^p d, w_\alpha) = R(c, w_\alpha) + R(d, w_\alpha)$

Using the induction hypothesis on c, d we get

$$= \tfrac{\mathbf{1}}{\mathbf{2}}(R(c, w_{\alpha_0}) + R(c, w_{\alpha_1})) + \tfrac{\mathbf{1}}{\mathbf{2}}(R(d, w_{\alpha_0}) + R(d, w_{\alpha_1}))$$

We can use additive commutativity and distributivity to get

$$= \tfrac{\mathbf{1}}{\mathbf{2}} \cdot (R(c, w_{\alpha_0}) + R(d, w_{\alpha_0}) + R(c, w_{\alpha_1}) + R(d, w_{\alpha_1}))$$
$$= \tfrac{\mathbf{1}}{\mathbf{2}} \cdot (R(c \vee^p d, w_{\alpha_0}) + R(c \vee^p d, w_{\alpha_1}))$$

Extended Frege can handle the bounded steps in each case of the inductive step. See appendix B for details on how the arithemtic can be handled by eFrege. □

Lemma 51. *For complete assignment α, we can prove using* eFrege *in the structure of the POG that $R(v, w_\alpha) = \mathbb{1}_v(\alpha)$.*

Proof. Again we show the base and inductive cases involve polynomially many basic steps (which each can be simulated by eFrege).

Variable leaf: Let the variable leaf be x_i. $R(x_i, w_\alpha) = \mathbf{1}$ or $R(x_i, w_\alpha) = \mathbf{0}$ since $|\alpha| = |X|$ and the value is determined entirely by $\mathbb{1}_{x_i}(\alpha)$.

Negation: Suppose $R(c, w_\alpha) = \mathbf{1}$ then by the induction hypothesis α satisfies c, so α falsifies $\neg c$ and $R(\neg c, w_\alpha) = \mathbf{1} - R(c, w_\alpha) = \mathbf{0}$.

Similarly, it can be derived for $R(c, w_\alpha) = \mathbf{0}$ that $R(\neg c, w_\alpha) = \mathbf{1}$.

Partition Conjunction: Let C be the set of child nodes for $\wedge^p$. If there is some $c^* \in C$ such that α falsifies c^* then by the induction hypothesis $R(c^*, w_\alpha) = \mathbf{0}$. Then we can prove $\prod_{c\in C} R(c, w_\alpha) = \mathbf{0}$ which is the formula for $R(\bigwedge^p_{c\in C} c, w_\alpha)$. Also, α must falsify $\bigwedge^p_{c\in C} c$ as it falsifies c^*.

In the other case, if no $c \in C$ is falsified, α satisfies all of C (we can state and prove this formally as a disjunction). Here, $R(c, w_\alpha) = 1$ for all $c \in C$ and $\prod_{c\in C} R(c, w_\alpha) = \mathbf{1}$. Also, α must satisfy $\bigwedge^p_{c\in C} c$ as it satisfies all $c \in C$.

Partition Disjunction: Let the child nodes of $\vee^p$ be c, d. If α falsifies both c and d then by induction hypothesis, $R(c, w_\alpha) = R(d, w_\alpha) = \mathbf{0}$. By adding these we get $\mathbf{0} = R(c, w_\alpha) + R(d, w_\alpha) = R(c \vee^p d, w_\alpha)$. Also, α must falsify $c \vee^p d$ as it falsifies both c, d.

Suppose α satisfies c, we can prove that α falsifies d using the Resolution proof δ included in the CPOG proof (and this is simulated by eFrege). Hence, $R(c, w_\alpha) = \mathbf{1}$, $R(d, w_\alpha) = \mathbf{0}$. Then $R(c \vee^p d, w_\alpha) = \mathbf{1} + \mathbf{0} = \mathbf{1}$. This can be repeated for when α satisfies d by using left identity instead of right identity. □

Lemma 52. *CPOG has short eFrege proofs of $\theta(\alpha) = \mathbb{1}_\phi(\alpha)$, when α is a complete assignment, and $\theta(\alpha) = \theta(\alpha_0) + \theta(\alpha_1)$, when α is strictly initial and partial.*

Proof. We define $\theta^i(\alpha) = \mathbf{2^i} \cdot R(r, w_\alpha)$. Using Lemma 50 we can show for the root node r that $R(r, w_\alpha) = \tfrac{\mathbf{1}}{\mathbf{2}} \cdot (R(r, w_{\alpha_0}) + R(r, w_{\alpha_1}))$ when α is initial and strictly partial. This proves the second property of restriction.

Using Lemma 51 we can show that $R(r, \alpha) = \mathbb{1}_r(\alpha)$ when α is complete. Hence $\theta^0(\alpha) = \mathbb{1}_r(\alpha)$. Here for the first property of restriction, we still need to prove that $\mathbb{1}_r(\alpha) = \mathbb{1}_\phi(\alpha)$. For this, we use the Resolution proofs for $r \leftrightarrow \phi$ in the CPOG proof. Since eFrege p-simulates Resolution, these are easily converted to show $\mathbb{1}_r(\alpha) = \mathbb{1}_\phi(\alpha)$. □

This proves Step 5 of our simulation technique for CPOG. Therefore from our simulation technique part 1 and 2, we have the following.

Theorem 53. CLIP+DRAT *p-simulates* CPOG.

In [3], the authors prove that CPOG is strictly stronger than the other existing proof systems (i.e. MICE, KCPS(#SAT)). Therefore we have the following.

Corollary 54. CLIP+DRAT *p-simulates* MICE *and* KCPS(#SAT).

7 Exponential Improvement on Existing #SAT proof systems

In this section, we give easy CLIP+eFrege proofs for hard formulas of existing proof systems. Below, in Corollary 56, we give easy CLIP+eFrege proofs of some unsatisfiable formulas which are hard in MICE and KCPS(#SAT). Next, we give easy proofs of XOR-PAIRS in CLIP+eFrege system (Theorem 65). These formulas were previously proven to be hard for MICE [5, Theorem 23]. Later, we give an easy CLIP^{NP} proof for the Symm$_n$ formulas which are known to be hard for dec-DNNFs [34, Theorem 10.3.8][1, Corollary 3.8], hence hard for both the MICE′ and KCPS(#SAT) systems.

7.1 UNSAT formulas

Let us briefly discuss about unsatisfiable formulas. That is, CNF formulas for which the model counts are $\mathbf{0}$. In [3], it has been shown that for unsatisfiable formulas, KCPS(#SAT) is p-equivalent to regular Resolution [3, Proposition 5.1] and MICE is p-equivalent to Resolution [3, Proposition 5.3]. In this paper, we observed the following for the unsatisfiable CNF formulas:

Proposition 55. *For unsatisfiable formulas ϕ, if ϕ has a short* eFrege *proof of unsatisfiability, then ϕ has a short* CLIP+eFrege *proof.*

Proof. For a unsatisfiable CNF ϕ, assume that it has an easy eFrege-proof of unsatisfiability. We can have an easy CLIP+eFrege proof of ϕ as follows: The cumulator ξ for ϕ is a trivial circuit that only outputs '$\mathbf{0}$' for any input. For any two consecutive assignments i.e $\beta_2 = \beta_1 + \mathbf{1}$, the inductive statement of lip encodes that $\xi(\beta_2) = \xi(\beta_1) + \mathbb{1}_\phi(\beta_2)$. Therefore, the eFrege proof of lip statement needs only the unsatisfiability proof of ϕ (i.e. $\mathbb{1}_\phi(\beta_2) = \mathbf{0}$). □

This gives more separation results for unsatisfiable formulas which are hard for Resolution but easy for eFrege. That is, we have the following:

Corollary 56. *The unsatisfiable formulas, PHP, clique-color and Random Parity have short proofs [16, 9, 15] in* CLIP+eFrege *but are hard [24, 28, 14] for* MICE *and* KCPS(#SAT).

Note that the Clique-coloring principle [28, Definition 7.1] is well studied in proof complexity. Informally, it encodes that if a graph G has a clique of size k, then G needs at least k colors. PHP is the famous Pigeon hole principle which encodes that if there are n pigeons and $n-1$ holes, at least one hole has more than one pigeon in it. Random Parity formulas are contradictions expressing both the parity and non-parity on a set of variables.

7.2 XOR-PAIRS

Consider the family XOR-PAIRS defined below.

Definition 57 (XOR-PAIRS [5])**.** *Let $X = \{x_1, \dots x_n\}$ and $Z = \{z_{1,1}, z_{1,2} \dots, z_{n,n-1}, z_{n,n}\}$.*
$C^1_{ij} = (x_i \vee x_j \vee \bar{z}_{ij}), C^2_{ij} = (\bar{x}_i \vee x_j \vee z_{ij}), C^3_{ij} = (x_i \vee \bar{x}_j \vee z_{ij}), C^4_{ij} = (\bar{x}_i \vee \bar{x}_j \vee \bar{z}_{ij})$
$\phi(X, Z)$ contains $C^1_{ij}, C^2_{ij}, C^3_{ij}, C^4_{ij}$ for $i, j \in [n]$.

The models of XOR-PAIRS are the assignments where $z_{i,j} = (x_i \oplus x_j)$ for all $i, j \in [n]$. Hence, $\#_{\text{models}}(\text{XOR-PAIRS}) = 2^n$. The family XOR-PAIRS is hard for proof system MICE′ [5, Theorem 23]. We will show in Theorem 65 that these formulas are easy in CLIP+eFrege.

Definition 58. *Fix an input length n, and let γ and δ be vectors of n variables. For pairs of individual variables a, b, use $a = b$ to denote $(\neg a \vee b) \wedge (\neg b \vee a)$. We can encode polynomial size propositional circuits: $L(\gamma, \delta)$, that denotes $num(\gamma) < num(\delta)$.*

We define the following gates. $L_n(\gamma, \delta) := (\neg\gamma_0 \wedge \delta_0)$. For $1 < i \leq n$, $L_i(\gamma, \delta) := (\neg\gamma_i \wedge \delta_i) \vee ((\gamma_i = \delta_i) \wedge L_{i-1}(\gamma, \delta))$. $L(\gamma, \delta) := L_1(\gamma, \delta)$.

Lemma 59. *Let γ and δ be vectors of n variables*

There is a short eFrege *proof of $L(\gamma, \delta) \to \bigvee_{i\geq 1}^{i\leq n} \overline{\gamma}_i$.*

Proof. **Induction hypothesis:** $L_j(\gamma, \delta) \to \bigvee_{i\leq j}^{i\geq 1} \overline{\gamma}_i$.
Base Case: $j = n$ and $L_n(\gamma, \delta) \to \overline{\gamma}_n \wedge \delta_n$ so $L_j(\gamma, \delta) \to \overline{\gamma}_j$.
Inductive Step: $L_j(\gamma, \delta) \to (\overline{\gamma}_j \wedge \delta_j) \vee L_{j+1}$ so $L_j(\gamma, \delta) \to \overline{\gamma}_j \vee \bigvee_{i>j}^{i\leq n} \overline{\gamma}_i$ via I.H. □

Lemma 60. *Let γ and δ be vectors of n variables. Recall Definitions 4 and 58. We have linear* eFrege *proofs of $L(\gamma, \delta) \vee L(\delta, \gamma) \vee E(\delta, \gamma)$.*

Proof. For pairs of individual variables a, b, we use $a < b$ to denote $\neg a \wedge b$ and $a = b$ to denote $(\neg a \vee b) \wedge (\neg b \wedge a)$. Also for brevity, denote the functions $L_i(\gamma, \delta)$ as $L_i^{\gamma,\delta}$ (see Definition 58), and $E_i(\gamma, \delta)$ as $E_i^{\gamma,\delta}$ (see Definition 4).
Induction hypothesis: $L_i^{\gamma,\delta} \vee L_i^{\delta,\gamma} \vee E_i^{\gamma,\delta}$ has a $O(n - i)$-size eFrege proof.
Base Case: $\neg L_n^{\gamma,\delta} \to (\gamma_n = \delta_n) \vee (\delta_n < \gamma_n)$ and hence $L_n^{\gamma,\delta} \vee L_n^{\delta,\gamma} \vee E_n^{\gamma,\delta}$
Inductive Step: $\neg L_i^{\gamma,\delta} \to ((\gamma_i = \delta_i) \vee (\delta_i < \gamma_i)) \wedge (\neg(\gamma_i = \delta_i) \vee (\neg L_{i+1}^{\gamma,\delta}))$. We can distribute this out: $\neg L_i^{\gamma,\delta} \to ((\gamma_i = \delta_i) \wedge (\neg L_{i+1}^{\gamma,\delta})) \vee ((\delta_i < \gamma_i) \wedge (\neg(\gamma_i = \delta_i)) \vee ((\delta_i < \gamma_i) \wedge (\neg L_{i+1}^{\gamma,\delta}))$. In fact $(\delta_i < \gamma_i)$ is sufficient for $L_i^{\delta,\gamma}$, so we simplify to $\neg L_i^{\gamma,\delta} \to ((\gamma_i = \delta_i) \wedge (\neg L_{i+1}^{\gamma,\delta})) \vee (\delta_i < \gamma_i)$.

Using the induction hypothesis we get $\neg L_i^{\gamma,\delta} \to ((\gamma_i = \delta_i) \wedge (E_{i+1}^{\gamma,\delta} \vee L_{i+1}^{\delta,\gamma})) \vee (\delta_i < \gamma_i)$ and this distributes to $\neg L_i^{\gamma,\delta} \to ((\gamma_i = \delta_i) \wedge E_{i+1}^{\gamma,\delta}) \vee ((\gamma_i = \delta_i) \wedge L_{i+1}^{\delta,\gamma})) \vee (\delta_i < \gamma_i)$. Essentially this is the $L_i^{\gamma,\delta} \vee L_i^{\delta,\gamma} \vee E_i^{\gamma,\delta}$.

□

Lemma 61. *Let γ and δ be vectors of n variables. We have linear* eFrege *proofs of $(L(\gamma, \delta) \vee E(\delta, \gamma)) \to \overline{L}(\delta, \gamma)$. Also for brevity, denote the functions $L_i(\gamma, \delta)$ as $L_i^{\gamma,\delta}$ (see Definition 58), and $E_i(\gamma, \delta)$ as $E_i^{\delta,\gamma}$ (see Definition 4).*

Proof. For pairs of individual variables a, b, we use $a < b$ to denote $\neg a \wedge b$ and $a = b$ to denote $(\neg a \vee b) \wedge (\neg b \wedge a)$. Also for brevity, denote the functions $L_i(\gamma, \delta)$ as $L_i^{\gamma,\delta}$ (see Definition 58), and $E_i(\gamma, \delta)$ as $E_i^{\gamma,\delta}$ (see Definition 4).
Induction hypothesis: $(L_j^{\gamma,\delta} \vee E_j^{\delta,\gamma}) \to \overline{L}_j^{\delta,\gamma}$
Base Case: $(j = n)$ $L_n^{\delta,\gamma} \to (\overline{\delta}_n \wedge \gamma_n)$ and $(\overline{\delta}_n \wedge \gamma_n) \to \overline{E}_n^{\delta,\gamma} \wedge \overline{L}_n^{\gamma,\delta}$
Inductive Step: Study the definition of $L_j^{\delta,\gamma}$, $E_j^{\delta,\gamma} \vee L_j^{\gamma,\delta}$ contradicts $(\overline{\delta}_j \wedge \gamma_j)$. So we check the $(\delta_j = \gamma_j) \wedge L_{j+1}^{\delta,\gamma}$ part. $(\delta_j = \gamma_j) \wedge L_j^{\gamma,\delta}$ forces $L_{j+1}^{\gamma,\delta}$ to be true, and $E_j^{\delta,\gamma}$ forces $E_{j+1}^{\delta,\gamma}$ to true. Since $E_{j+1}^{\delta,\gamma} \vee L_{j+1}^{\gamma,\delta}$ implies $\overline{L}_{j+1}^{\delta,\gamma}$, $(\delta_j = \gamma_j) \wedge L_{j+1}^{\delta,\gamma}$ is also refuted by $E_j^{\delta,\gamma} \vee L_j^{\gamma,\delta}$. □

Lemma 62. *Let α, β and γ be vectors of n variables. Recall Definitions 4 and 58. We have polynomial size proofs of*

$T(\beta, \alpha) \to L(\beta, \alpha) \wedge (\overline{L}(\alpha, \gamma) \vee \overline{L}(\gamma, \beta))$

Proof. For each $i : 1 \leq i \leq n$, we prove the following tautology: $T(\beta,\alpha) \to ((\overline{\alpha}_i \wedge \bigwedge_{j>i}^{j\leq n} \alpha_j) \to (\beta_i \wedge \bigwedge_{k<i}^{k\geq 1}(\alpha_k = \beta_k)))$ which follows from the definition of T.

We can first prove $(\overline{\alpha}_i \wedge \beta_i \wedge \bigwedge_{k<i}^{k\geq 1}(\alpha_k = \beta_k) \to \bigwedge_{j\leq i}^{j\geq 1} L_j(\alpha,\beta))$ in a linear size proof. $T(\beta,\alpha)$ implies $\bigvee_{i\geq 1}^{i\leq n} \overline{\alpha}_i$ so we get $L_1(\alpha,\beta)$.

We can inductively prove $\bigwedge_{j\geq i}^{j\leq n} \alpha_j \to \overline{L}_i(\alpha,\gamma)_i$. Likewise with $\bigwedge_{j\geq i}^{j\leq n} \overline{\beta}_j \to \overline{L}_i(\gamma,\beta)_i$. Therefore both $\overline{\alpha}_i \wedge \bigwedge_{j>i}^{j\leq n} \alpha_j \wedge L_i(\alpha,\gamma)_i \to \gamma_i$, and $\beta_i \wedge \bigwedge_{j>i}^{j\leq n} \overline{\beta}_j \wedge L_i(\gamma,\beta) \to \overline{\gamma}_i$ have linear size proofs in $n-i$. Since $\overline{\alpha}_i \wedge \bigwedge_{j>i}^{j\leq n} \alpha_j$ and $\beta_i \wedge \bigwedge_{j>i}^{j\leq n} \overline{\beta}_j$ are equivalent under $T(\beta,\alpha)$ and $\bigvee_{1\leq i\leq n} \overline{\alpha}_i \wedge \bigwedge_{j>i}^{j\leq n} \alpha_j$ is implied by $T(\beta,\alpha)$. We only have to show one more thing, that for any k, $\alpha_k = \beta_k \wedge (\overline{L}_{k+1}(\alpha,\gamma) \vee \overline{L}_{k+1}(\gamma,\beta)) \to (\overline{L}_k(\alpha,\gamma) \vee \overline{L}_k(\gamma,\beta))$ which comes out the definition of L. Assembling this all together we get $T(\beta,\alpha) \to L(\beta,\alpha) \wedge (\overline{L}(\alpha,\gamma) \vee \overline{L}(\gamma,\beta))$. □

Lemma 63. *Let α,β,γ and δ be vectors of n variables. Recall Definition 4. We have short* eFrege *proof of $T(\alpha,\beta) \wedge T(\gamma,\delta) \to (E(\alpha,\gamma) = E(\beta,\delta))$*

Proof. Suppose $T(\alpha,\beta) \wedge T(\gamma,\delta) \wedge E(\alpha,\gamma)$ then we can prove for each bit $\beta_i = (\alpha_i \oplus \bigwedge_{j\geq i} \alpha_j) = (\gamma_i \oplus \bigwedge_{j\geq i} \gamma_j) = \delta_i$ and this can be assembled into $E(\beta,\delta)$.

Now for the converse, suppose $T(\alpha,\beta) \wedge T(\gamma,\delta) \wedge E(\beta,\delta)$. We can prove by induction starting with $(\alpha_n = \gamma_n)$ that each bit is the same.

□

Lemma 64. *Let α,β and γ be vectors of n variables. Recall Definitions 4 and 58. We have polynomial size proofs of*

$L(\alpha,\beta) \to ((E(\beta,\gamma) \vee L(\beta,\gamma)) \to L(\alpha,\gamma))$.

Proof. For brevity, denote the functions $L_i(\gamma,\delta)$ as $L_i^{\gamma,\delta}$ (see Definition 58), and $E_i(\gamma,\delta)$ as $E_i^{\gamma,\delta}$ (see Definition 4).

Induction Hypothesis: $L_j^{\alpha,\beta} \wedge (L_j^{\beta,\gamma} \vee E_j^{\beta,\gamma}) \to L_j^{\alpha,\gamma}$.

Base Case: $(\overline{\alpha}_n \wedge \beta_n)$ contradicts $(\overline{\beta}_n \wedge \gamma_n)$ and if $\beta_n = \gamma_n$ then $(\overline{\alpha}_n \wedge \gamma_n)$

Inductive Step: $(\overline{\alpha}_j \wedge \beta_j)$ contradicts $(\overline{\beta}_j \wedge \gamma_j)$, so if $(\overline{\alpha}_j \wedge \beta_j)$ and $L_j^{\beta,\gamma} \vee E_j^{\beta,\gamma}$ are both true then $\beta_j = \gamma_j$ and thus $(\overline{\alpha}_j \wedge \gamma_j)$.

Now suppose instead $(\alpha_j = \beta_j) \wedge L_{j+1}^{\alpha,\beta}$. If $(\overline{\beta}_j \wedge \gamma_j)$ then $(\overline{\alpha}_j \wedge \gamma_j)$, otherwise the only other way to have $L_j^{\beta,\gamma} \vee E_j^{\beta,\gamma}$ true is to have both $\beta_j = \gamma_j$ and $L_{j+1}^{\beta,\gamma} \vee E_{j+1}^{\beta,\gamma}$ which proves $L_{j+1}^{\alpha,\gamma}$. Since $\alpha_j = \gamma_j$ we get $L_{j+1}^{\alpha,\gamma}$. □

Theorem 65. CLIP+eFrege *has short proofs of XOR-PAIRS*

Proof. First we fix that all Z-bits are less significant than all X-bits, otherwise the cumulator function is affected by the variable ordering. We begin by arguing that the cumulative function for XOR-PAIRS is easy to compute. This comes from the fact the truth function itself behaves in a way that makes it amenable to counting, it only ever increases by one, once for each complete assignment to X. There is a function $p : 2^X \to 2^Z$ that maps the binary assignment α on X to the unique assignment in Z such that $\phi(\alpha, p(\alpha))$ for every α.

We can construct a multi-output circuit P (a sequence of circuits $P_{i,j}$ for $i,j \in \{|X|\}$) for p, easily through $O(Z)$ many gates.

$$P_{i,j}(X) = (x_i \vee x_j) \wedge (\overline{x}_i \vee \overline{x}_j)$$

We then express the cumulative function in a cumulator circuit that we will use for CLIP.

$$\xi(\alpha,\beta) = \begin{cases} \alpha & \beta < P(\alpha) \\ \alpha+1 & \beta \geq P(\alpha) \end{cases}$$

Note that since $\xi(\alpha,\beta)$ outputs in binary we can actually express each digit as a Boolean circuit: $\xi(\alpha,\beta)_i = (L(\beta,P(\alpha)) \wedge \alpha_i) \vee (\overline{L}(\beta,P(\alpha)) \wedge (\alpha_i \oplus \bigwedge_{j\leq n}^{i<j} \alpha_j))$

Now we have to argue why the remaining propositional proof is easy for eFrege. Omitting num^{-1}, the propositional formula ($\mathsf{lip}(\xi)$ from Definition 5) is:
$\big(\overline{E}(A_X,0) \vee \overline{E}(A_Z,0) \vee \phi(A_X,A_Z) \vee E(\xi(A_X,A_Z),0)\big)\wedge$
$\big(\overline{E}(A_X,0) \vee \overline{E}(A_Z,0) \vee \overline{\phi(A_X,A_Z)} \vee T(\xi(A_X,A_Z),0)\big)\wedge$
$\big(\overline{E}(B_X,A_X) \vee \overline{T}(B_Z,A_Z) \vee \phi(B_X,B_Z) \vee E(\xi(B_X,B_Z),\xi(A_X,A_Z))\big)\wedge$
$\big(\overline{E}(B_X,A_X) \vee \overline{T}(B_Z,A_Z) \vee \overline{\phi(B_X,B_Z)} \vee T(\xi(B_X,B_Z),\xi(A_X,A_Z))\big)\wedge$
$\big(\overline{T}(B_X,A_X) \vee \overline{E}(B_Z,0) \vee \overline{E}(A_Z,2^{|Z|}-1) \vee \phi(B_X,B_Z) \vee E(\xi(B_X,B_Z),\xi(A_X,A_Z))\big)\wedge$
$\big(\overline{T}(B_X,A_X) \vee \overline{E}(B_Z,0) \vee \overline{E}(A_Z,2^{|Z|}-1) \vee \overline{\phi(B_X,B_Z)} \vee T(\xi(B_X,B_Z),\xi(A_X,A_Z))\big)\wedge$
$\big(\overline{E}(A_X,2^{|X|}-1) \vee \overline{E}(A_Z,2^{|Z|}-1) \vee E(\xi(A),k)\big)$

This is basically a number of tautological implications we have to show individually The idea is to break each implication into a number of cases. Case analysis is typically easy for eFrege as it is just resolving with a disjunction of possibilities. This is where we use Lemma 60 which gives us the disjunction of possibilities.

Base case: If $A_X = 0$, $P(A_X)$ always evaluates to 0. If A_Z is also 0, $\phi(A_X,A_Z)$ evaluates to true, while $L(A_Z,P(A_X))$ evaluates to false (because of strictness). This makes $\xi(\alpha,\beta)$ evaluate to the integer 1 (in other words $\xi(\alpha,\beta)_i = 1$ if and only if $i = n$). Each of these evaluations are shown in eFrege through the extension clauses. These will satisfy the two disjunctions that use the base case.

Inductive Step: Here we firstly argue that $\phi(B_X,B_Z) \leftrightarrow E(B_Z,P(B_X))$ has a short eFrege proof. We show that for each pair i,j the four clauses are implied by $(x_i \vee x_j) \wedge (\overline{x}_i \vee \overline{x}_j) \leftrightarrow z_{i,j}$. And then we show the four clauses show the truth table for$(x_i \vee x_j) \wedge (\overline{x}_i \vee \overline{x}_j) \leftrightarrow z_{i,j}$. The proof size is linear.

If $B_X = A_X$ and $B_Z = A_Z + 1$, we use Lemma 60 to make 3 cases.

1. Let $B_Z = P(B_X)$, we can get a short eFrege proof that $L(B_Z,P(B_X))$ is false (Lemma 60) and that $L(A_Z,P(A_X))$ is true (Lemma 64). And thus a proof of $T(\xi(B_X,B_Z),B_X)$ and $E(\xi(A_X,A_Z),A_X)$, we use $B_X = A_X$ to show $T(\xi(B_X,B_Z),\xi(A_X,A_Z))$. $\phi(B_X,B_Z) \leftrightarrow E(B_Z,P(B_X))$ is a proven tautology.

2. Let $B_Z < P(B_X)$, we know immediately that $L(B_Z,P(B_X))$ is true and also a short proof that $L(A_Z,P(A_X))$ is true. And thus a proof that $E(\xi(B_X,B_Z),B_X)$ and $E(\xi(A_X,A_Z),A_X)$, we use $B_X = A_X$ to show $E(\xi(B_X,B_Z),\xi(A_X,A_Z))$. $\phi(B_X,B_Z)$ falls into provable contradiction with $L(B_Z,P(B_X))$ by showing a bit must be different.

3. Let $B_Z > P(B_X)$, we can get a short eFrege proof that $L(B_Z,P(B_X))$ is false (Lemma 60) and that $L(A_Z,P(A_X))$ is false (Lemma 62). And thus a proof that $T(\xi(B_X,B_Z) = B_X + 1)$ and $T(\xi(A_X,A_Z) = A_X + 1)$, we use $B_X = A_X$ to show $E(\xi(B_X,B_Z),\xi(A_X,A_Z))$. $\phi(B_X,B_Z)$ falls into provable contradiction with $L(P(B_X),B_Z)$.

Now consider $||B_X = A_X + 1||$, $||B_Z = 0||$ and $||A_Z = 2^{|Z|} - 1||$. Part of the trichotomy is impossible. We can prove $L(P(B_X),B_Z)$ fails when $||B_Z = 0||$. For the remaining cases we firstly prove that $\neg||2^{|Z|} - 1 < P(A_X)||$ which is proven from the fact that one digit must be 0 to be less than. Therefore $\xi(A_X,A_Z) = A_X + 1$ in both cases.

1. Let $B_Z = P(B_X)$ then we can get a short eFrege proof that $L(B_Z, P(B_X))$ is false and so $\xi(B_X, B_Z) = B_X + 1 = A_X + 1 + 1 = \xi(A_X, A_Z) + 1$. We can use the T function and Lemma 63 to find an equality proof here. $\phi(B_X, B_Z) \leftrightarrow E(B_Z, P(B_X))$ is a tautology,

2. Let $L(B_Z, P(B_X))$ be true so $\xi(B_X, B_Z) = B_X = A_X + 1 = \xi(A_X, A_Z)$. $\phi(B_X, B_Z)$ falls into provable contradiction with $L(B_Z, P(B_X))$.

For the final case, we once again use $\neg||2^{|Z|} - 1 < P(A_X)||$, hence $\xi(2^{|X|} - 1, 2^{|Z|} - 1) = 2^{|X|} - 1 + 1 = 2^{|X|}$.

□

7.3 Permutation matrices

Consider the Boolean function f_{Symm}[2] which verifies if a Boolean $[n \times n]$ matrix is a permutation matrix, i.e. each row and each column contains exactly one entry 1. The satisfying models, i.e. $f^{-1}_{\mathsf{Symm}}(1)$ are all $n!$ row-permutations of the n-by-n identity matrix. This function is hard to represent as a dec-DNNF [34, Theorem 10.3.8][1, Corollary 3.8] and hence it is hard to count models of the CNF representing f_{Symm} (Definition 66) with the existing static proof system KCPS(#SAT) [12]. Since extraction of dec-DNNFs is easy for MICE′ [6], this is also a lower bound for MICE′. We show that these formulas are easy in CLIP (Theorem 67).

Definition 66. *Let every cell in an $[n \times n]$ matrix be a variable $x_{i,j}$ where i is the row number and j is the column number. The CNF Symm_n with $\mathcal{O}(n^3)$ clauses is defined as follows:*
$C^1_{i,j} := x_{i,j} \rightarrow (\overline{x}_{i,1} \wedge ... \wedge \overline{x}_{i,j-1} \wedge \overline{x}_{i,j+1} \wedge ... \wedge \overline{x}_{i,n} \wedge \overline{x}_{1,j} \wedge ... \wedge \overline{x}_{i-1,j} \wedge \overline{x}_{i+1,j} \wedge ... \wedge \overline{x}_{n,j})$,
$C^2_i := (x_{i,1} \vee x_{i,2} \vee ... \vee x_{i,n})$, $C^3_i := (x_{1,i} \vee x_{2,i} \vee ... \vee x_{n,i})$
$\mathsf{Symm}_n := \bigwedge_{i,j \in [n]} C^1_{i,j} \wedge \bigwedge_{i \in [n]} C^2_i \wedge \bigwedge_{i \in [n]} C^3_i$.

Theorem 67. CLIP^{NP} *has short proofs of Symm_n.*

Proof. In order to show that Symm_n is easy for CLIP^{NP}, we first present an Algorithm which given Symm_n and a partial assignment α returns the $C_{models}(\alpha)$ (see Algorithm 2). Observe that any row-permutation of an identity matrix of size $[n \times n]$ is a model of Symm_n. Algorithm 2 maintains a ascending list (i.e. pos_rows array in Algorithm 2) of all rows of an identity matrix of size $[n \times n]$. Given any total assignment α, the Algorithm breaks it row-wise as $J_1, ..., J_n$. Starting from J_1, it compares J_i with every row in pos_rows. If $J_i >$ a row in pos_rows (i.e. row in Algorithm 2), we can include all $(n - i)!$ additional models which have the same row as the i^{th} row in them. If any row-vector (i.e. row) is equal to the J_i, we remove it from the pos_rows array and repeat the process for J_{i+1}. As we always maintain pos_rows array in the ascending order, we can stop when any J_i is $<$ a row and return the cumulative number of models. In the worst case, J_1 is greater than $n - 1$ rows and equal to n^{th} row, similarly J_2 is greater than $n - 2$ rows and equal to $(n - 1)^{th}$ row and so on, resulting in $\mathcal{O}(n^2)$ complexity. A polynomial size cumulator for Symm_n can be built based on Algorithm 2. □

We provide Example 68 to illustrate the working of Algorithm 2.

Example 68. *Consider a $[3 \times 3]$ Boolean matrix (i.e. $n = 3$). The pos_rows array for this matrix will be $:= \{1, 2, 4\}$. Consider an assignment $num(\alpha_1) = 511$, i.e. the last assignment where*

[2] In [34], this function is denoted as PERM. To avoid confusion with the #P-complete problem 'permanent', we denote it here as f_{Symm} for 'Symmetric group' of the identity matrix along with its permutations.

Algorithm 2 Algorithm to compute $C_{models}(\mathsf{Symm}_n, \alpha)$

Require: $num(\alpha) < 2^{n^2}$, ordering of variables is in row-major order.
(i.e $x_{1,1}, ..., x_{1,n}, x_{2,1}, ..., x_{2,n}, ...$)
function CUMULATIVE-PERM-MODELS(assignment α, int n)
int $J_1, ..., J_n = 0$
for $i \in [n]$ **do**
for $j \in [n]$ **do**
$J_i = J_i * 10 + \alpha(x_{i,j})$ /*J_i is the $num(\alpha$ restricted to the i^{th} row)*/
end for
end for
int pos_rows $= \{1, 2, 4, 8, ..., 2^{n-1}\}$ /* possible rows in any model*/
int $i = 1$, int $models = 0$, int flag $= 0$
while flag=0 **do**
for row $\in$ pos_rows **do**
if row $< J_i$ **then** /*include all models having row as the i^{th} row in them*/
$models = models + (n - i)!$
else if row $> J_i$ **then** /*break the loop and return the number of models*/
flag= 1
break-loop
else/*if J_i=row, remove row from array and start for-loop again with J_{i+1}*/
$i++$, flag= 0
pos_rows = pos_rows $\setminus$ row
break-loop
end if
flag= 1
end for
end while
return *models*
end function

every variable is set to 1*, in this case the Algorithm 2 should return the total* $\#_{models}(\mathsf{Symm}_3) = C_{models}(\mathsf{Symm}_3, 511) = 6$*. In the algorithm,* $J_1 = 7$ *which is greater than all elements of the* pos_rows *array, therefore the algorithm will return* $2! + 2! + 2! = 6$ *in just* $\mathcal{O}(n)$ *steps.*

Consider another assignment $num(\alpha_2) = 154$*. In the algorithm,* $J_1 = 2$, $J_2 = 3$, $J_3 = 2$*. First* $J_1 >$pos_rows[0] *hence the current model count is* $2!$*. Next,* $J_1 =$pos_rows[1] *hence the updated* pos_rows *array is* $:= \{1, 4\}$*. Next,* $J_2 >$pos_rows[0] *hence current model count is* $2! + 1!$*. Lastly* $J_2 <$pos_rows[1] *therefore the algorithm stops here and returns the* $C_{models}(\mathsf{Symm}_3, 154) = 3$*. To cross-verify the three models before* α_2 *are* $84, 98$ *and* 140*.*

8 Potential for new Benchmarks

The p-simulation of CPOG provides a method to extract CLIP+DRAT proofs indirectly from model counting programs, but the number of arithmetic lemmas may bloat the simulation.

One alternative is to avoid the second part of the p-simulation that constructs the proof and directly find the proof through a SAT solver. Without having implemented this ourselves it is unclear how well this works in practice. We suggest that the next step would be to create UNSAT

benchmarks from the lip formulas for known families of model counting problems, such as XOR-pairs using the cumulator we construct in Section 7.

8.1 DQBF Benchmarks

QBF (Quantified Boolean Formulas) extend propositional logic with a quantifier prefix over the propositional variables. The range of the quantifiers are the Boolean constants $\{0, 1\}$. DQBF (Dependency Quantified Boolean Formulas) extends QBF by forgoing the linear quantifier ordering. In S-form a DQBF has a number of universal variables $\forall u_1 \dots u_p$ and a number of existential variables written in the form $\exists x_1(D_1) \dots x_q(D_q)$. The sets D_i are subsets of the universal variables that x_i depends on.

We can formulate the lip statement without a specific cumulator in mind, by asking if there exists a cumulator.

$\forall X_1 \forall X_2 \exists Z_1(X_1) \exists Z_1(X_1)$
$(X_1 = X_2) \rightarrow (Z_1 = Z_2) \wedge$
$E(X_1, num^{-1}(\mathbf{0})) \rightarrow \big((\overline{\phi(X_1)} \rightarrow E(Z_1, num^{-1}(\mathbf{0}))) \wedge (\phi(X_1) \rightarrow T(Z_1, num^{-1}(\mathbf{0})))\big) \wedge$
$T(X_2, X_1) \rightarrow \big((\overline{\phi(X_2)} \rightarrow E(Z_2, Z_1))) \wedge (\phi(X_2) \rightarrow T(Z_2, Z_1))\big) \wedge$
$E(X_1, num^{-1}(2^{|vars(\phi)|} - 1)) \rightarrow E(Z_1, num^{-1}(k))$

When we ask whether Z_1 variables exists, we are asking if there is an output identical to a correct cumulator for the X_1. Z_2 is the same for X_2 variables, and Z_1 and Z_2 are forced to have the same Skolem functions by the new constraint.

We use DQBF because we do not want Z_1 to "cheat" by changing depending on X_2 variables (which could be used to trivially satisfy the statement).

New benchmarks for DQBF are hard to come by. A certifying DQBF solver (like `Pedant` [31]) may output circuits for the Skolem functions. Unfortunately, for modern DQBF solvers it is very likely to be hard in that case except in the smallest of examples.

9 Conclusion

We have introduced the CLIP framework for propositional model counting. We have demonstrated the advantages CLIP has by having an unrestricted underlying circuit format. Our approach here has been theoretical and no version of CLIP has been implemented.

The main checking task in CLIP proofs can use existing tools in SAT such as DRAT-trim [35]. We have given a p-simulation of all other #SAT proof systems, in theory this can be used to extract CLIP+eFrege (or CLIP+DRAT) proofs from #SAT solvers. However the number of arithmetic lemmas may make the complete programming of the extractor a difficult task. It could be compensated with assistance from a certifying SAT solver.

Future work should take into account weighted and projected model counting.

Acknowledgements

This work was supported by FWF ESPRIT grant no. ESP 197. Additional thanks to Olaf Beyersdorff, Tim Hoffmann, Luc Nicolas Spachmann as well as the anonymous reviewers for the discussions on this topic.

References

[1] Paul Beame, Jerry Li, Sudeepa Roy, and Dan Suciu. Lower bounds for exact model counting and applications in probabilistic databases. In Ann E. Nicholson and Padhraic Smyth, editors, *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11-15, 2013*. AUAI Press, 2013. URL: `https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2366&proceeding_id=29`.

[2] Olaf Beyersdorff, Ilario Bonacina, Leroy Chew, and Jan Pich. Frege systems for quantified Boolean logic. *J. ACM*, 67(2), April 2020.

[3] Olaf Beyersdorff, Johannes K. Fichte, Markus Hecher, Tim Hoffmann, and Kaspar Kasche. The Relative Strength of #SAT Proof Systems. In *27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024)*, Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[4] Olaf Beyersdorff, Luke Hinde, and Ján Pich. Reasons for hardness in QBF proof systems. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, December 11-15, 2017, Kanpur, India*, volume 93 of *LIPIcs*, pages 14:1–14:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[5] Olaf Beyersdorff, Tim Hoffmann, and Luc Nicolas Spachmann. Proof complexity of propositional model counting. In Meena Mahajan and Friedrich Slivovsky, editors, *26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy*, volume 271 of *LIPIcs*, pages 2:1–2:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: `https://doi.org/10.4230/LIPIcs.SAT.2023.2`, `doi:10.4230/LIPICS.SAT.2023.2`.

[6] Olaf Beyersdorff, Tim Hoffmann, and Luc Nicolas Spachmann. Proof complexity of propositional model counting. *Electron. Colloquium Comput. Complex.*, pages TR24–030, 2024. URL: `https://eccc.weizmann.ac.il/report/2024/030`.

[7] Randal E Bryant, Wojciech Nawrocki, Jeremy Avigad, and Marijn J. H. Heule. Certified knowledge compilation with application to verified model counting. In *26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

[8] Randal E. Bryant, Wojciech Nawrocki, Jeremy Avigad, and Marijn J. H. Heule. Supplement to certified knowledge compilation with application to verified model counting. 2023. URL: `https://zenodo.org/records/7966174`.

[9] Sam Buss and Neil Thapen. DRAT and propagation redundancy proofs without new variables. *Log. Methods Comput. Sci.*, 17(2), 2021.

[10] Samuel R. Buss. Towards NP-P via proof complexity and search. *Ann. Pure Appl. Log.*, 163(7):906–917, 2012.

[11] Florent Capelli. *Structural restriction of CNF-formulas: application to model counting and knowledge compilation*. PhD thesis, UFR de Mathématiques - Université Paris Cité, 2016.

[12] Florent Capelli. Knowledge compilation languages as proof systems. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 90–99. Springer, 2019. doi:10.1007/978-3-030-24258-9_6.

[13] Hubie Chen. Proof complexity modulo the polynomial hierarchy: Understanding alternation as a source of hardness. In *ICALP 2016*, pages 94:1–94:14, 2016.

[14] Leroy Chew, Alexis de Colnet, Friedrich Slivovsky, and Stefan Szeider. Hardness of random reordered encodings of parity for resolution and CDCL. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 7978–7986. AAAI Press, 2024. URL: https://doi.org/10.1609/aaai.v38i8.28635, doi:10.1609/AAAI.V38I8.28635.

[15] Leroy Chew and Marijn J. H. Heule. Sorting parity encodings by reusing variables. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2020. doi:10.1007/978-3-030-51825-7_1.

[16] Stephen A Cook. A short proof of the pigeon hole principle using extended resolution. *Acm Sigact News*, 8(4):28–32, 1976.

[17] Stephen A Cook and Robert A Reckhow. The relative efficiency of propositional proof systems. In *Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook*, pages 173–192. ACM, 2023.

[18] William J. Cook, Collette R. Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discret. Appl. Math.*, 18(1):25–38, 1987.

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

[20] Peter M. Fenwick. A new data structure for cumulative frequency tables. *Softw. Pract. Exp.*, 24(3):327–336, 1994. URL: https://doi.org/10.1002/spe.4380240306, doi:10.1002/SPE.4380240306.

[21] Johannes K Fichte, Markus Hecher, and Valentin Roland. Proofs for propositional model counting. In *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[22] Gottlob Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache der reinen Denkens, Halle.* Lubrecht & Cramer, 1879. English translation in: from Frege to Gödel, a source book in mathematical logic (J. van Heijenoord editor), Harvard University Press, Cambridge 1967.

[23] Joshua A. Grochow and Toniann Pitassi. Circuit complexity, proof complexity, and polynomial identity testing: The ideal proof system. *J. ACM*, 65(6), nov 2018. doi:10.1145/3230742.

[24] Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985.

[25] Jinbo Huang and Adnan Darwiche. DPLL with a trace: From SAT to knowledge compilation. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 156–162. Professional Book Center, 2005. URL: http://ijcai.org/Proceedings/05/Papers/0876.pdf.

[26] Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory.* Cambridge University Press, New York, NY, USA, 1995.

[27] Benjamin Kiesl, Adrián Rebola-Pardo, and Marijn JH Heule. Extended resolution simulates DRAT. In *Automated Reasoning: 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, pages 516–531. Springer, 2018.

[28] Jan Krajícek. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *J. Symb. Log.*, 62(2):457–486, 1997.

[29] Meena Mahajan and Gaurav Sood. QBF merge resolution is powerful but unnatural. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 22:1–22:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: https://doi.org/10.4230/LIPIcs.SAT.2022.22, doi:10.4230/LIPICS.SAT.2022.22.

[30] Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[31] Franz-Xaver Reichl and Friedrich Slivovsky. Pedant: A Certifying DQBF Solver. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*, volume 236 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:10, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.SAT.2022.20, doi:10.4230/LIPIcs.SAT.2022.20.

[32] John Alan Robinson. Theorem-proving on the computer. *Journal of the ACM*, 10(2):163–174, 1963.

[33] Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. doi:10.1137/0220053.

[34] Ingo Wegener. *Branching Programs and Binary Decision Diagrams.* SIAM, 2000. URL: http://ls2-www.cs.uni-dortmund.de/monographs/bdd/.

[35] Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, 2014.

Appendix

A Algorithm from Section 4.1

We provide a more detailed version of Algorithm 1.

Algorithm 1 Fenwick tree [20] based algorithm to find $D(0, J)$

Require: $J < 2^n$
  function FENWICK-ASSIGNMENTS(int J, int n)
    int $\alpha := \{\}$, dash:= $\{\}$
    int $indx \leftarrow J + 1$ /*assignments $\in [0, 2^n - 1]$ but the Fenwick tree handles $[1, 2^n]$ */
    while $indx > 0$ **do**
      parent = $indx - (indx \ \& \ -indx)$ /*& is the bit-wise AND operator*/
      α.append(parent)
      dash.append($log(indx - \text{parent})$) /*records number of variables to forget from the corresponding total assignment α*/
      $indx \leftarrow$ parent
    end while
    int $len = |\alpha|, D[len, n]$
    /*D is a 2d array where every row is a partial assignment of length n*/
    for $i \in [len]$ **do**
      for $j \in [n - dash[i] - 1]$ **do**
        if binary($\alpha[i]$)[j] == 1 **then** /* if j^{th} bit in binary representation($\alpha[i]$)= 1 */
          $D[i, j] := 2$
        else
          $D[i, j] := 1$ /* if j^{th} bit in binary representation($\alpha[i]$)= 0 */
        end if
      end for
      for $j \in [n - dash[i], n - 1]$ **do**
        $D[i, j] := 0$ /* if j^{th} variable is not included in partial assignment */
      end for
    end for
  /* if $\alpha_i(x_j) = 1 \rightarrow D[i, j] = 2$, if $\alpha_i(x_j) = 0 \rightarrow D[i, j] = 1$, if $x_j \notin vars(\alpha_i) \rightarrow D[i, j] = 0$*/
  return D
  end function

B Short proofs on basic arithmetic properties

Extended Frege can handle basic tautologies on its own. In fact, it is unlikely that a given well-known identity would turn out the be a lower bound for eFrege. It could be said the short proofs of basic arithmetic properties of gates are academic folklore. For the sake clarifying to the reader that we have not obscured any difficult lemmas into this, we give an overview of the construction of each of the proofs.

We take for granted that eFrege can deal with a constant number of cases, in fact we can see this from a simulation of truth table. eFrege can also handle proofs by induction formalising a polynomial number of inductive steps into extension variables.

Definition 69 (Signed Binary number)**.** *A signed binary number X contains a sign bit s_x and a number of binary bits $x_{n-1}, x_{n-2} \dots x_1, x_0, x_{-1} \dots x_{-m}$. s_x is 1 if X is positive, allowing for straightforward addition. Binary fractions are used for CPOG.*

Definition 70 (Addition Operator)**.** *We use $\oplus$ for exclusive or.*

We first define the carry bits

$$C_i = \begin{cases} X_i \wedge Y_i & i = -m \\ (X_i \wedge Y_i) \vee (X_i \wedge C_{i-1}) \vee (Y_i \wedge C_{i-1}) & -m < i < n \end{cases}$$

And then the bits of the sum

$$A_i = \begin{cases} X_i \oplus Y_i & i = -m \\ X_i \oplus Y_i \oplus C_{i-1} & -m < i < n \end{cases}$$

Where $X + Y = A = (A_i)_{i>-m}^{i<n}$

For signed integers we define $s_{a+b} = (s_A \wedge s_A) \vee (s_A \wedge C_n) \vee (s_B \wedge C_n)$. It works like the nth carry bit.

Lemma 71 (Additive Associativity)**.** *Given vectors of propositional variables X, Y, Z there are short* eFrege *proofs of $||X + (Y + Z) = (X + Y) + Z||$.*

Proof. Let V be defined as $(X + Y)$ and W be defined as $(Y + Z)$. Let P be defined as $(V + Z)$ and Q be defined as $(X + W)$ Let C^V, C^W, C^P, C^Q be the carry functions for each of the respective sums.

We show $C_i^W \oplus C_i^Q = C_i^V \oplus C_i^P$ by induction on i. The inductive step can be expanded into a tautology of seven variables: $C_{i+1}^V, C_{i+1}^P, C_{i+1}^W, C_{i+1}^Q, X_i, Y_i, Z_i$. Since this can be proven by truth table, we have short eFrege proofs.

□

Lemma 72 (Additive Commutativity)**.** *Given vectors of propositional variables A, B there are short* eFrege *proofs of $||A + B = B + A||$.*

Proof. We consider both $A + B$ and $B + A$ and show each output bit and carry bit are equal inductively. Each induction step requires a constant bounded number of variables. □

Lemma 73 (Additive Identity)**.** *Given any vector of propositional variables A there are short* eFrege *proofs of $||\mathbf{0} + A = A||$.*

Proof. We show each bit in $\mathbf{0} + A$ is equal to the bit in the summand A and each carry bit is 0. Proving the carry bit is 0 takes $|A|$ many induction steps, but each induction step only involves a constant number of variables. □

Definition 74 (Additive Inverse)**.** *The additive inverse of a is defined as*

$$(-a)_i = \begin{cases} a_i & i = -m \\ a_i \oplus c_{i-1} & i > -m \end{cases}$$

The definition $c_i = \begin{cases} (a_i \wedge (-a)_i) \vee (a_i \wedge c_{i-1}) \vee ((-a)_i \wedge c_{i-1}) & i = -m \\ a_i \oplus c_i & i > -m \end{cases}$

$s_{(-a)} = \neg s_a$

Lemma 75 (Additive Inverse)**.** *Gievn a vector of propositional variables A there are short* eFrege *proofs of $A + (-A) = \mathbf{0}$*

It turns out the carry bit used in the definition will be the same as the carry bit used when adding a value to its inverse.

Proof. We prove inductively that if two bits to be summed have parity 1, the previous carry bit is 1. Likewise if the carry bit is present the two summand bits have parity 1. In other words $a_i \oplus (-a)_i \oplus c_{i-1} = 0$. We can break each inductive step (and the base case) into a finite case analysis, thus short proofs.

Only the sign bit is allowed to be different but then the final carry bit changes that to 1. □

Definition 76 (Multiplication Operator)**.** *The partial product p_i is a multiplied by $b_i \cdot 2^i$. We define $p_{i,j} = b_i \wedge a_{j-i}$.*

The product $a \cdot b = (-1)^{s_a + s_b} \cdot \sum_i p_i$

Lemma 77 (Multiply by $\mathbf{0}$)**.** *For any vector of propositional terms A. There are short* eFrege *proofs that $||A \cdot \mathbf{0} = \mathbf{0}||$*

Proof. First we show show every partial product is $\mathbf{0}$. We can easily prove that each partial product bit is 0 by definition of $\wedge$. To finalise we then use additive identity to get $\mathbf{0}$ □

Lemma 78 (Multiplicative Identity)**.**

Proof. We prove every partial product is zero except for p_0, which we then prove is identical to the multiplicand using the definition of $\wedge$. We use the additive identity law and associativity and commutativity of addition to get the multiplicand back. □

Lemma 79 (Multiply by powers of 2)**.** *Let A be a vector of propositional variables. There are short* eFrege *proofs that $A \cdot 2^i = B$ where $B_j = 0$ when $j < i - m$ and $B_j = A_{j-i}$ otherwise and 2^i is defined as a constant vector.*

Sketch Proof. Similar to identity except we first show only p_i is potentially non zero via multiplication using the definition of $\wedge$. The defintion of $\wedge$ also tells us the bits of p_i. $p_i = B$ through additive identity. □

Lemma 80 (Binary Decomposition)**.** *Every number can be shown to be equal to a sum of powers of two: $||A = \sum_{0 \leq i}^{i<n} \mathbb{1}_{A_i} \cdot 2^i||$ where 2^i is defined as a constant vector.*

Proof. Outside the "diagonal" (p_{ii}) every bit is 0 and we prove as such. We show that all carry bits for every sum is zero. Again we do this via induction. Then show ith bit of the sum is 1 if and only if there is a 1 in the ith bit of the summand 2^i. Likewise show the ith bit of the sum is 0 if and only if the ith bit of the ith summand is 0. □

Lemma 81 (Weak Left Distributivity)**.** *Let A, B be vectors of propositional variables. There are short* eFrege *proofs of* $||A \cdot B = \sum_{-m \leq i}^{i<n} A_i \cdot (\mathbb{1}_{B_i} \cdot 2^i)||$

Proof. Let p_j be the jth partial product of $A \cdot B$ and let q_j^i be the jth partial product of $A_i \cdot (\mathbb{1}_{B_i} \cdot 2^i)$.

$q_j^i = \mathbf{0}$ when $j \neq i$ and $q_j^j = p_j$. Therefore using additive identity, commutativity and associativity $A \cdot B$ is the same sum as $\sum_{-m \leq i}^{i<n} A_i \cdot (\mathbb{1}_{B_i} \cdot 2^i)$

□

Lemma 82 (Weak Right Distributivity)**.** *Let A, B be vectors of propositional variables. There are short* eFrege *proofs of* $||A \cdot B = \sum_{-m \leq i}^{i<n} (\mathbb{1}_{A_i} \cdot 2^i) \cdot B_i||$

Proof. let $p_{j,k}^i$ be the kth bit of the jth partial product of $(\mathbb{1}_{A_i} \cdot 2^i) \cdot B$ $p_{j,k}^i$ is only 1 when $k = j - i$ and $A_i = 1$ and $B_j = 1$.

Let q_j be the jth partial product of $A \cdot B$, $q_{j,k} = 1$ only when $A_{k-j} = 1$ and $B_j = 1$. One can observe that this means that $q_j = \sum_i p_{j,k}^i$ and we can prove that in eFrege by showing the carry bits will never be 1.

Thus we can use associativity and commutatitivity of addition to show that $A \cdot B = \sum_j q_j = \sum_j \sum_i p_j^i = \sum_i (\mathbb{1}_{A_i} \cdot 2^i) \cdot B$

□

By using the full expansion we can prove the other properties of arithmetic.

Lemma 83 (Commutativity)**.** *Let A, B be vectors of propositional variables. There are short* eFrege *proofs of* $||A \cdot B = B \cdot A||$

Proof. We perform a full expansion on both sides and show equality through additive associativity and commutativity. □

Lemma 84 (Associativity)**.** *Let A, B, C be vectors of propositional variables. There are short* eFrege *proofs of* $||A \cdot (B \cdot C) = (A \cdot B) \cdot C||$

Proof. We perform a full cubic expansion, and then use additive associativity and commutativity to show the sums are the same. □

Lemma 85 (Distributivity)**.** *Let A, B, C be vectors of propositional variables. There are short* eFrege *proofs of* $||A \cdot (B + C) = A \cdot B + A \cdot C||$

Proof. We show the partial products sum to the same value by decomposing them readjusting their order. □

Lemma 86. $\frac{1}{2}(a + a) = a$

Proof. We prove inductively that $a + a$ creates a left shift. Its then relatively easy to show that right multiplication with $\frac{1}{2}$ is a right shift and we use commutativity. □

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il