Electronic Colloquium on Computational Complexity, 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

April 2, 2024

Abstract

In this paper we present a new proof system framework CLIP (Cumulation Linear Induction Proposition) for propositional model counting. A CLIP proof firstly involves a 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 allows us to use existing checking formats from propositional logic, especially strong proof systems. Despite model counting being a harder problem than unsatisfiability, we find that CLIP only has lower bounds from its propositional proof system or if $\mathsf{P}^{\#\mathsf{P}}$ is not contained in P/poly, similar to results in QBF proof complexity [6]. From a proof complexity point of view we find it is exponentially stronger than existing proof systems MICE [25] and KCPS(#SAT) [14] for propositional model counting.

1 Introduction

In mathematics, a *proof by induction* is a convenient and concise way of proving a fact for a large or even infinite domain. The main driving force in a proof by induction is the proof of the inductive step. Informally we can think of the induction step as universally quantifying over the domain; whether each instance of the induction hypothesis is implied by the previous instance of the induction hypothesis. Nonetheless an induction step is often used to simplify a seemingly complicated theorem. We want to use this idea to develop a framework that takes model counting problem and projects them into propositional tautologies representing an inductive step.

Propositional model counting (#SAT) is an extension of SAT-solving that counts the number of models (satisfying assignments) of a propositional formula. Unsatifiability is just the special case when the number of models is zero. For unsatisfiability, we have a number of powerful proof systems, including the checking format DRAT (Deletion Resolution Asymmetric Tautology) [49], Frege and Extended Frege systems and the Ideal Proof System (IPS)[29] which uses an unrestricted circuit as a static certificate for an inconsistent set of polynomial equations. For IPS the existence of lower bounds directly relates to circuit complexity. Where IPS only has super-polynomial lower bounds if $\mathsf{VNP} \neq \mathsf{VP}$. We should not focus too hard on these specific complexity classes here, but instead note that one of best ways to demonstrate strength in a proof system is to make lower bounds conditional on another complexity problem.

In the past few decade work has been undertaken to create efficient #SAT algorithms [11, 4, 42, 45, 23, 36]. Some work has also been undertaken for proof systems for #SAT. KCPS(#SAT) [14] is a static model counting proof system that uses the checkability of dec-DNNFs (decision Decomposable Negation Normal Form), but retains lower bounds from dec-DNNFs. The MICE proof system [25] is

ISSN 1433-8092

a dynamic system that argues from axioms involving a single model to more complicated expressions with many models, however it was shown that MICE has an exponential lower bound involving XOR-pairs [9].

For a number of applications, utilizing propositional model counting alongside proofs of correctness would be clearly beneficial. A proof may be a direct part of the application, i.e. if we were to use model counting for part of a hard combinatorics problem. Proofs could be used as part of a verified systems, i.e. in planning. Since propositional model counting can be used directly in Bayesian inference, verification can add some trust into systems that use probabilistic methods.

We could perhaps compare progress in proofs for propositional model counting to that in Quantified Boolean formulas (QBFs). QBF are another extension of SAT, that uses both existential and universal variables. The main thing we need to know about QBF for this paper is that we have many strong proof systems for QBF already [6, 30]. In fact some proof systems have conditional optimality, where a QBF proof size lower bound exists if and only if either a propositional proof size lower bound exists or an open circuit complexity conjecture is true [6].

1.1 Our Contribution: Certifying #SAT via Propositional Logic

Propositional model counting is #P-complete [39], this is an interesting position in computational complexity; Toda's Theorem [46] means #SAT is hard for every level of the polynomial hierarchy, while also in functional PSPACE [1, p. 344].

Our initial approach was that because #SAT is in PSPACE we can translate the problem into a series of QBF queries, and simply present a series of QBF proofs (or just one proof of a conjunction of many QBFs). As long as the reduction was correct this would be sound, and we can use efficient QBF proof systems. $\mathsf{eFrege}+\forall\mathsf{red}$ for example only has a lower bound if eFrege has a lower bound or $\mathsf{PSPACE} \nsubseteq \mathsf{P/poly}$ [6]. These are both long-standing open problems that we do not expect to solve by simply attempting the current #SAT lower bounds for MICE and KCPS(#SAT).

However, while QBFs and the proof complexity of $\mathsf{eFrege}+\forall\mathsf{red}$ make a good point of comparison, the actual use of QBFs turned out to be unnecessary. If we observe the PSPACE algorithm for #SAT we can skip unnecessary quantifying reasoning to find a proof system even more intuitive.

We can iteratively count the number of models of a propositional CNF formula Φ by evaluating it 2^n-times (for n variables) once in every possible complete assignment to its variables, and keeping a cumulative count [39, Section 18.2]. Apart from storing the cumulative count, and the current position (both binary integers each using n-bits), we can free up memory after each assignment and retain polynomial space. This cumulative count effectively records the state of the counting machine at the critical points of its runtime. We know this algorithm is correct by inductively knowing the cumulative count is correct. Our proposal is to use a proof system of two parts:

1. A non-deterministically guessed Boolean circuit which when given the current assignment in the runtime it returns the cumulative count of models.

2. A purely propositional proof by induction that said guessed circuit is correct.

The resulting system $\mathsf{CLIP}+\mathcal{P}$ (Cumulative Linear Induction Proposition, Definition 5) is not necessarily a static system, because its propositional part ($\mathcal{P}$) may be dynamic. We can consider part of the proof i.e. the choice of circuit for the cumulative count as static. If we allow the propositional part to be a static proof system i.e. CLIP+IPS we get a fully static system. There is also the idea to forego fitting Cook and Reckhow's [22] notion of a proof system and use an NP-oracle instead of a propositional proof.

If we either use a powerful propositional proof system $\mathsf{CLIP}+\mathcal{P}$ or an NP-oracle $\mathsf{CLIP}^{\mathsf{NP}}$ we can show using Fenwick assignment trees [24] that we have a framework that can simulate any model counting proof systems that is closed under restrictions, including KCPS(#SAT) (Theorems 5,6) and MICE (Theorems 8,9). Furthermore we show that hard formulas for either system are easy for CLIP proofs (Theorems 10,11).

Organisation of the paper: In Section 3 we define our CLIP framework for propositional model counting proof systems. In Section 4 we give the arguments for how CLIP can simulate existing proofs systems KCPS(#SAT) and MICE. In Section 5 we show that CLIP can handle formulas that are hard for existing systems. We conclude this paper in Section 6 with some discussions and possible directions for future research.

1.2 Related Work

Propositional model counting is the problem of counting the *exact* number of models of the given CNF formula Φ. Other related problems in literature are weighted [10] and projected model counting [2]. In weighted model counting, every total assignment is assigned a weight and the output is the sum of weights of all models. Projected model counting asks to count models with respect to a given set of projection variables, where models that are identical when restricted to the projection variables count as only one model.

CPOG (Certified Partitioned-Operation Graphs): An important proof system CPOG was defined in [12] to overcome the weakness (as discussed in [12, p.3]) of KCPS(#SAT) framework when used as a solver [15]. CPOG works with any practical solver outputting a dec-DNNF by using a external proof checker. This CPOG is inspired by DRAT [49] and Extended Resolution [47] and works with partial ordered graphs. It works on both standard and weighted model counting.

Approximate Model Counting: Since exact model counting is hard, finding an approximation of the count with high probability is interesting. It turns out that the problem of approximating the model-count has several practical importance as well, for example, in fields like probabilistic reasoning [41]. This leads to developing several practical solvers in the literature [27, 37]. For further details on approximate model counting, interested readers are referred to the book chapters [28, 16].

SAT and QBF proof complexity: In propositional logic, we have weak proof systems like Resolution [40], which capture solving techniques and stronger proof systems like DRAT [49] and Extended Resolution, which are used as a checking format. Extensive work has been done investigating the relationship between propositional proof systems [44]. Quantified Boolean Formulas are the canonical PSPACE-complete language and there are again proof systems such as QU-Resolution [48], Long distance Q-Resolution [50], $\forall$Exp+Res [33] and more powerful proof systems like QRAT [30] and eFrege + $\forall$red [6]. Again, emerging works on QBF proof complexity have uncovered the relationships between proof systems [21, 7, 3, 43]

2 Preliminaries

For a Boolean variable x, its literals can be x and $\neg x$. 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 $\perp$. $vars(C)$ is a set of all variables in C and $vars(\Phi) = \bigcup_{C \in \Phi} vars(C)$. We say Φ is a trivial CNF if it either contains an empty clause or all its clauses contain the constant literal 1.

2.1 Assignments

Given a CNF Φ on n-variables ($X = \{x_1, ..., x_n\}$), a total assignment α to $vars(\Phi)$ is a function which maps every variable $\in X$ to 0/1. α is said to be a satisfying assignment (model) of Φ if every clause $C \in \Phi$ has a literal ℓ such that $\alpha(\ell) = 1$ ($\alpha(\bar{\ell}) = 1 - \alpha(\ell)$). $\Phi(\alpha) = 1$ if α is a model of Φ and $\Phi(\alpha) = 0$ otherwise. We denote the total number of models of a CNF Φ as $\#_{\text{models}}(\Phi)$. The model counting problem (#SAT) is the problem of computing $\#_{\text{models}}(\Phi)$ for a given CNF Φ. #SAT is the canonical complete problem for the function class #P. The class of languages decidable in polynomial time with an oracle to #SAT are denoted by $\mathrm{P}^{\#\mathrm{P}}$.

A partial assignment is a function which maps a subset of X to 0/1. For an assignment α, $\mathit{vars}(\alpha)$ denotes the set of variables assigned in α. Let α, β be two partial assignments to variables X, they are said to be consistent ($\alpha \simeq \beta$) if $\mathit{vars}(\alpha) \cap \mathit{vars}(\beta)$ are assigned the same 0/1-value in both α and β, otherwise they are said to be inconsistent ($\alpha \not\simeq \beta$). A partial assignment α when appended with 0/1 assignment to the variables $X \setminus \mathit{vars}(\alpha)$, will form a total assignment α' this is said to be extending a partial assignment. 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 individually. 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.

We can fix an ordering among variables in X that determines the binary encoding. In a vector of Boolean variables (or constants) $x_1, \dots, x_n$ we treat x_n as the least significant bit and x_1 as the most significant bit (MSB). We denote $\mathit{num}(\alpha)$ as the integer corresponding to any total assignment α of $\mathit{vars}(\Phi)$. To be precise, $\mathit{num}(\alpha) := \Sigma_{i \in [n]} \alpha(x_i) * 2^{n-i}$ for any total assignment α. Observe that if total assignment $\alpha := \{x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 1\}$ for $n = 4$, then $\mathit{num}(\alpha) = 7$. We can view assignments as integers or as a set or as a function interchangeably. In places where we refer to arithmetic statements coded in propositional logic, we use a binary encoding for numbers. For the reverse, we use the notation $|| \dots ||$ to denote the conversion of arithmetic statements into propositional logic. We denote $[J]$ to denote numbers $\{1, ..., J-1, J\}$ and $[J_1, J_2]$ to denote numbers $\{J_1, J_1 + 1, ..., J_2 - 1, J_2\}$.

2.2 Proof Systems

A *proof system* [22] is a polynomial-time functions 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 say f *simulates* g when there is a polynomial function p, such that for every g-proof π_1 there is an f-proof π_2, such that $g(\pi_1) = f(\pi_2)$ and $|\pi_2| \leq p(|\pi_1|)$. We say that 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.

Definition 1 (Closure under restrictions [38])**.** *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(Φ), 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.*

This is the stricter definition which we use in this paper. A more weaker version which states that the π' need not be extracted from π is also used in the literature.

Propositional Proof Systems: The language UNSAT is the set of CNF formulas which have 0 satisfying assignments. Propositional proof systems map refutations to formulas from the language UNSAT.

Resolution is arguably the most studied propositional proof system. It has the rule $\frac{(C \vee x) \quad (D \vee \bar{x})}{(C \cup D)}$ 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 $\rho := \Phi \mathrel{\left|\frac{\quad}{Res}\right.} \bot$). 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 [26] 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 [22]. Figure 1 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 1: A Frege system for connectives $\to, 0, 1$

Extended Frege (eFrege) [22] 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. When showing that eFrege has short proofs we can use the fact it can simulate many different proof systems such as Resolution, Cutting planes, Truth tables. In fact eFrege can simulate any proof system as long as there is a short proof of the reflection principle of said proof system [32].
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 [17, 8].
The existing #SAT proof systems MICE$'$ and KCPS(#SAT) are defined in Section 4.

2.3 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. P/poly is the class of Boolean functions computed by polynomial-sized circuit families.

A 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 $\in \Phi$.

An arithmetic circuit is a DAG where every input gate (with indegree 0) is labelled by a variable or a constant ($\in$ fixed field F) and every other gate is labelled with either '+' (for addition) or '$*$' (for multiplication). Arithmetic circuits inherently represent polynomials. 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. We denote the Boolean XOR gate with $\oplus$ in the paper.

3 Cumulation Linear Induction Proposition (CLIP) Proof Framework

In this section, we define 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 circuit ξ (denoted as a *cumulator*) which outputs the total number of models of Φ from total assignment 0 to a given total assignment $num(\alpha)$ (denoted as $C_{models}(\Phi, \alpha)$, Definition 2). Clearly, when $num(\alpha) = 2^{n-1}$, $C_{models}(\Phi, \alpha) = \#_{\text{models}}(\Phi)$. 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 $vars(\Phi)$. For any assignment α to $vars(\Phi)$, 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 0 to assignment $num(\alpha)$. 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.*

Definition 3 (Cumulator)**.** *A cumulator for a CNF Φ over n variables is a Boolean circuit $\xi(\alpha)$ which takes as input an assignment α to $vars(\Phi)$ (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 $num(\alpha) = 2^{|n|} - 1$), $\xi(\alpha)$ 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 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 $num(\alpha) = 0$, we need to verify that the following is satisfied: $(\Phi(\alpha)$ & $(\xi(\alpha) = 1))$ OR $(\overline{\Phi(\alpha)}$ & $(\xi(\alpha) = 0))$. This covers the case that if the first assignment is a model for Φ then the cumulator should return 1, else a 0.

For the inductive step when $num(\alpha) = num(\beta) + 1$, the following should be satisfied $(\Phi(\alpha)$ & $(\xi(\alpha) = \xi(\beta) + 1))$ OR $(\overline{\Phi(\alpha)}$ & $(\xi(\alpha) = \xi(\beta))$. 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, cumulator should output the same number under both assignments.

For the final case when $num(\alpha) = 2^{|vars(\Phi)|} - 1$, 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 ($||$ $||$ encloses the arithmetic comparisons needed):

$||num(\alpha) = 0|| \to \big((\Phi(\alpha) \wedge ||\xi(\alpha) = 1||) \vee (\neg\Phi(\alpha) \wedge ||\xi(\alpha) = 0||)\big)$
$\wedge ||num(\alpha) = num(\beta) + 1|| \to \big((\Phi(\alpha) \wedge ||\xi(\alpha) = \xi(\beta) + 1||) \vee (\neg\Phi(\alpha) \wedge ||\xi(\alpha) = \xi(\beta)||)\big)$
$\wedge ||num(\alpha) = 2^{|vars(\Phi)|} - 1|| \to ||\xi(x) = k||$.

To convert this into a purely propositional statement, we need Boolean circuits to implement the arithmetic conditions $||x = y||$ and $||y = x + 1||$ for any integers x, y. We define polynomial sized Boolean circuits for the same as $E(x, y)$ and $T(y, x)$ 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 $(\neg a \vee b) \wedge (\neg b \wedge a)$. We can encode polynomial size propositional circuits:*

- *$E(\gamma, \delta)$, that denotes $num(\gamma) = num(\delta)$: $E_n(\gamma, \delta) := (\gamma_n = \delta_n)$. For $1 \leq i < n$, $E_i(\gamma, \delta) := (\gamma_i = \delta_i) \wedge E_{i+1}(\gamma, \delta)$. $E(\gamma, \delta) := E_1(\gamma, \delta)$.*
- *$T(\gamma, \delta)$, that denotes $num(\gamma) = num(\delta) + 1$. For $0 < i \leq n$ and accepting the empty conjunction as true, $S(\delta)_i := \neg(\delta_i = \bigwedge_{j>i}^{j\leq n} \delta_j)$. $T(\gamma, \delta) := E(\gamma, S(\delta)) \wedge \bigvee_{i\geq 1}^{i\leq n} \overline{\delta}_i$.*

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 statement* check(ξ)*. Let num map assignments to integers using the standard binary encoding, and num^{-1} be its inverse. Let $E(\gamma, \delta)$ be a circuit that verifies equality ($\gamma = \delta$), let $T(\gamma, \delta)$ denote a Boolean circuit that verifies if $num(\gamma) = num(\delta) + 1$. Let A and B be two disjoint copies of the variables in Φ. The following is a tautology in the variables of $A \cup B$:*

$$\begin{aligned}
&\mathit{check}(\xi) := \\
&E(A, num^{-1}(0)) \rightarrow \big((\overline{\Phi(A)} \rightarrow E(\xi(A), num^{-1}(0))) \wedge (\Phi(A) \rightarrow T(\xi(A), num^{-1}(0)))\big) \wedge \\
&T(B, A) \rightarrow \big((\overline{\Phi(B)} \rightarrow E(\xi(B), \xi(A))) \wedge (\Phi(B) \rightarrow T(\xi(B), \xi(A)))\big) \wedge \\
&E(A, num^{-1}(2^{|vars(\Phi)|} - 1)) \rightarrow E(\xi(A), num^{-1}(k))
\end{aligned}$$

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

Theorem 1. *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 $\overline{\text{check}(\xi)}$ from the above definition.

For polynomial time checkability, we perform 3 steps: 1) verify that ξ is indeed a circuit. 2) Using ξ, generate check(ξ) once again, to make sure it matches (where $\mathcal{P}$ does not accept circuits a canonical translation, i.e. Tseitinisation is needed). 3) verifying the $\mathcal{P}$ proof. $\square$

Theorem 2. CLIP$+$eFrege *has a super-polynomial lower bound only if* eFrege *has a super-polynomial lower bound or* $\mathsf{P}^{\#\mathsf{P}} \not\subseteq \mathsf{P/poly}$

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, check(ξ_n) is also polynomial size in Φ_n. Thus the family $(\text{check}(\xi_n))_{n\geq 0}$ is super-polynomial lower bound for eFrege. $\square$

We observe that if we use the powerful static proof system IPS (Ideal Proof System) [29] within the CLIP framework, we can relate lower bounds in the clip framework with the famous open problem in arithmetic circuit complexity. To be precise, we have the following.

Theorem 3. *If* CLIP*+IPS has a super-polynomial lower bound then* $\mathsf{VNP} \neq \mathsf{VP}$ *or* $\mathsf{P}^{\#\mathsf{P}} \not\subseteq \mathsf{P/poly}$

Proof. Suppose CLIP+IPS has a super-polynomial lower bound and $\mathsf{P}^{\#\mathsf{P}} \subset \mathsf{P/poly}$. Then as in the previous proof there is a family $(\mathsf{check}(\xi_n))_{n\geq 0}$ that is a super-polynomial lower bound for IPS. A super-polynomial lower bound for IPS implies $\mathsf{VNP} \neq \mathsf{VP}$ [29, Theorem 1.2]. □

The lesson here seems to be that circuits are a powerful way to compactly represent proofs.

Definition 6 (CLIP^{NP})**.** *The proof system* CLIP^{NP} *for #SAT is a cumulator* ξ *for CNF* Φ *along with its correctness encoded by the statement* $\mathsf{check}(\xi)$ *from Definition 5 and verified by an NP-oracle instead of a proof.*

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

4 CLIP framework Simulations of existing #SAT Proof Systems via Cumulator Extraction

In this section, we give a general framework of extracting efficiently a cumulator circuit from the existing propositional model counting proof systems. This helps in the simulations of these proof systems via the CLIP framework. We need the following definition for the same.

Definition 7 (Disjoint binary partial assignment cover ($\mathsf{PAC}(J_1,J_2)$))**.** *Let* J_1, J_2 *be some total assignments to variables* $X := x_1, ..., x_n$ *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 $X := \{x_1, x_2, x_3, x_4\}$, $J_1 := 5$ and $J_2 := 15$. One possible $\mathsf{PAC}(J_1,J_2) := \{\{x_1 = 1\}, \{x_1 = 0, x_2 = 1, x_3 = 1\}, \{x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 1\}\}$. Observe that the first partial assignment (i.e. $\{x_1 = 1\}$) is covering all assignments from $[8, 15]$. Similarly the second and third partial assignments are covering the assignments $[6, 7]$ and 5 respectively.

Let us now outline the general extraction technique.

Cumulator Extraction Technique: Let $\mathcal{P} \in \{\mathsf{MICE}', \mathsf{KCPS}(\#\mathsf{SAT})\}$ 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).
2. For any total assignment J to $vars(\Phi)$, find the set of non-overlapping partial assignments (to $vars(\Phi)$) which cover the entire assignment space from assignments $0 \rightarrow J$ (i.e $\mathsf{PAC}(0,J)$ see Definition 7).

   Using Fenwick's idea [24], it is easy to compute $\mathsf{PAC}(0,J)$ for any total assignment J, such that $|\mathsf{PAC}(0,J)| \leq n$ (Lemma 1).
3. For each partial assignment $\alpha \in \mathsf{PAC}(0,J)$, restrict π with α and consider the $\mathcal{P}$-proof π' of CNF $\Phi|_\alpha$. 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 $C_{models}(\Phi, J)$ and takes $\mathcal{O}(n.|\pi|)$ time.

We prove Step-1 of our extraction technique individually for both existing proof systems in Sections 4.1 and 4.2. Let us now focus on proving Step-2 of our extraction technique.

Lemma 1. *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 $[0, J]$ (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 [24] 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 $log(2^n) = n$. We provide Example 1 to illustrate the use of Algorithm 1. □

Algorithm 1 Fenwick tree [24] based algorithm to find PAC(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*/
          $\alpha$.append(parent)
          dash.append($log(indx -$ parent$)$)  /*records no. of variables to forget from $\alpha$*/
          $indx \leftarrow$ parent
      end while
  return $\alpha' =$ process($\alpha$, 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 of the fixed ordering of
  variables*/
  end function
```

Example 1. *Let $n = 5$ and $J = 22$. 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, dash are $\{22, 20, 16, 0\}$ and $\{0, 1, 2, 4\}$ respectively. The corresponding PAC(0,22) is the following set of partial assignments: $\Big\{\{x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1, x_5 = 0\}, \ \{x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0\}, \ \{x_1 = 1, x_2 = 0, x_3 = 0\}, \ \{x_1 = 0\}\Big\}$.

We handle Step-3, 4 of our extraction technique individually for proof systems KCPS(#SAT) and MICE′ in the Sections 4.1 and 4.2 respectively. This completes the extraction technique.

4.1 CLIP framework simulation of KCPS(#SAT)

In this section, we apply our extraction technique to the Knowledge Compilation based Proof System (KCPS(#SAT)). In [14], the authors define a static propositional model counting proof system based on the knowledge representation known as dec-DNNF(see Definition 8). First we redefine the KCPS(#SAT) propositional model counting proof system (Definition 10) from [14]. We then show that KCPS(#SAT) is closed under restrictions (Theorem 4), which is the first step in our extraction technique. Step-3, 4 of the extraction technique are proved in Lemma 2.

Definition 8 (dec-DNNF [31])**.** *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 [14, 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 $\#_{\text{models}}(D)$. As a result, we have the following.

Proposition 1. *[13, Proposition 1.57][14, 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 9 (cert-dec-DNNF [14])**.** *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, $C_s|_\alpha = \perp$. We denote by $F(D) := \bigwedge_{s \in 0\text{-}sinks(D)} C_s$.*

It is known that Proposition 1 holds even for cert-dec-DNNFs.

Definition 10 (KCPS(#SAT) [14])**.** *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 4. 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*(Φ) 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 $x \in \mathit{vars}(\alpha)$, 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 $x \notin \mathit{vars}(\alpha)$, 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 2. 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 $F(D') \subseteq \Phi|_\alpha$.

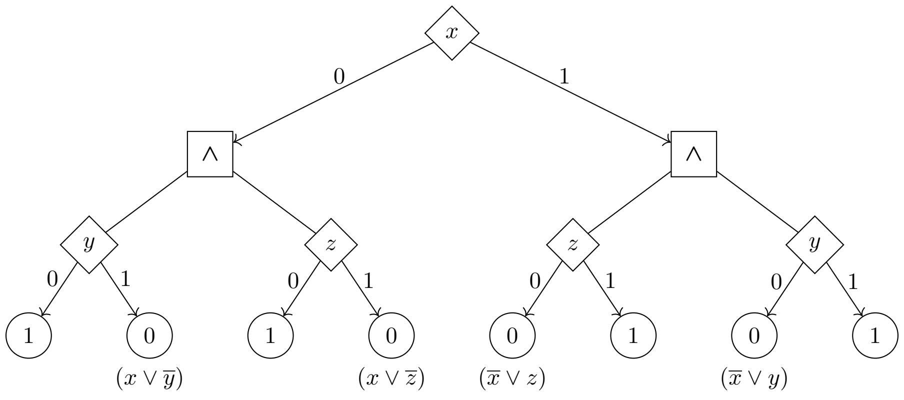
For the other direction, we need to show that $D' \to \Phi|_\alpha$: Since D is correct, we know that $D \to C$, for all $C \in \Phi$. That is, for all total assignments γ to *vars*(Φ), if $D(\gamma) = 1$ then $C|_\gamma = 1$. We need to prove the following:

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

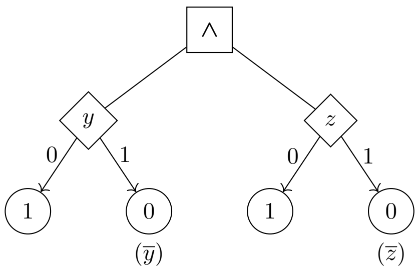
The easy case is when the partial assignment α satisfies all the clauses in Φ. That is, for every clause $C \in \Phi$, α sets at least one of its literals to 1. Then conclusion of equation 1 is always true and we are done.

For the remaining partial assignments α, assume for contradiction that equation 1 is not true. Then, there is a total assignment β for $\big(\mathit{vars}(\Phi) \setminus \mathit{vars}(\alpha)\big)$ 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*(Φ). 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. ☐

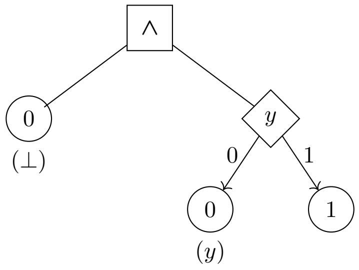
Example 2 ([14])**.** *Consider a simple example of a CNF computing $x = y = z$ as $\Phi := (\overline{x} \vee y) \wedge (\overline{y} \vee x) \wedge (\overline{x} \vee z) \wedge (\overline{z} \vee x)$. Φ has 2 models $x = y = z = 0$ and $x = y = z = 1$. A* KCPS(#SAT) *proof of Φ is shown in Figure 2a. The decision nodes are shown as diamond-shaped with the label of its*



(a) cert-dec-DNNF D for CNF Φ [14]



(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 2: cert-dec-DNNF used in Example 2

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 4, we build the cert-dec-DNNFD' for $\Phi|_{\alpha_1}$ as shown in Figure 2b. 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 4, we build the cert-dec-DNNF D'' for $\Phi|_{\alpha_2}$ as shown in Figure 2c. Note that $\#_{\text{models}}(D'') = \#_{\text{models}}(\Phi|_{\alpha_2}) = 0$.

Lemma 2. *Given a KCPS(#SAT) 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 cumulative number of models of Φ in $[0, J]$ (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 1, 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 4). For each such D', we use Proposition 1 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 1. *From* KCPS(#SAT) *proofs we can efficiently extract cumulator circuits.*

Theorem 5. CLIP^{NP} *simulates* KCPS(#SAT)*.*

Proof. From a KCPS(#SAT) proof we extract the cumulator circuit ξ in polynomial time. Therefore check(ξ) is polynomial size, and is true because of the correctness we have argued for Theorem 4 and Lemma 1. The NP-oracle finds check(ξ) in a single line. □

In Krajíček's book on *Bounded Arithmetic, Propositional Logic, and Complexity Theory* [32] 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 6. *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.

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

Extract(KCPS(#SAT)) is complete since it accepts any eFrege proof. Its soundness is shown through correctness of the cumulator extraction of Lemma 2. 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. 1. 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 [32]. *Extract*(KCPS(#SAT)) is simulated by eFrege + $\|\text{refl}(\mathit{Extract}(\mathsf{KCPS(\#SAT)}))\|$, 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. □

4.2 CLIP framework simulation of MICE′

In this section, we apply our extraction technique to MICE′ (Model-counting Induction by Claim Extension) [25, 9]. MICE′ is a propositional model counting proof system which is shown to be p-equivalent to MICE [25] but has simpler inference-rules. Hence extracting cumulator circuits from MICE′ proofs is sufficient. First we redefine the MICE′ (Definition 11) from [9]. We then show that MICE′ is closed under restrictions (Theorem 7). Step-3, 4 of the extraction technique are proved in Lemma 3.

Definition 11 (MICE′ [9])**.** *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 3.* 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:

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

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:

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

(E1) $F_1 \subseteq F$

(E2) $A|_{vars(F_1)} = A_1$

(E3) A satisfies $F \setminus F_1$

Figure 3: MICE′ derivation rules

In the proof of Theorem 7, we need the following definition.

Definition 12. *[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 7. 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: $(\emptyset,\ \emptyset,\ 1)$ by the axiom-rule, $(\Phi|_\alpha, \emptyset, 2^{|vars(\Phi|_\alpha)|})$ 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: $\overline{(\Phi|_\alpha,\ \emptyset,\ 0),\ \rho}$ 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 12) 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 $L'_i := (F_i|_\alpha, A_i \setminus \mathit{vars}(\alpha), c'_i) \in \pi'$ such that if L'_i is not invalid (as discussed above), then $c'_i = \#_{\text{models}}(F_i|_{\alpha \cup A_i \setminus \mathit{vars}(\alpha)})$.

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 $\overline{(F_i,\emptyset,0),\rho}$, $\rho := F_i \mid\!\frac{}{\mathit{Res}} \bot$ and as Resolution is closed under restrictions, $\rho|_\alpha := F_i|_\alpha \mid\!\frac{}{\mathit{Res}} \bot$. 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:

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

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

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

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

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

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

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

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

     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 $L'_j := (F_1|_\alpha,\ A_1 \setminus \mathit{vars}(\alpha),\ c'_1)$ 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 $L'_j := (F_1|_\alpha,\ A_1 \setminus \mathit{vars}(\alpha),\ c'_1), L'_k := (F_2|_\alpha,\ A_2 \setminus \mathit{vars}(\alpha),\ c'_2)$ in π' are valid. Then set

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

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

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

   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 $L'_{k_1} := (F|_\alpha,\ A_1 \setminus \mathit{vars}(\alpha),\ c'_1), ..., L'_{k_j} := (F|_\alpha,\ A_j \setminus \mathit{vars}(\alpha),\ c'_j)$) are invalid but $A \simeq \alpha$. Then set

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

   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 $\rho := (F \cup A \cup \{\overline{A_i} | i \in [n]\}) \vdash_{\mathit{Res}} \perp$ now because $A_1, ..., A_j \not\simeq \alpha$ restricting $\rho|_\alpha := (F|_\alpha \cup A \setminus \mathit{vars}(\alpha) \cup \{\overline{A_i} | i \in [j+1,n]\}) \vdash_{\mathit{Res}} \perp$ 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 3. *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* $\rho := (x \vee z), (\overline{x} \vee \overline{z}), (z), (\overline{z}), \perp\Big\}$.

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* $\rho|_\alpha := (z), \perp\Big\}$

We derive π' using the procedure defined in proof of Theorem 7. 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 $\pi' := \{(\emptyset,\ \emptyset,\ 1), (\Phi|_\alpha,\ y = z = 0,\ 1), (\Phi|_\alpha,\ \emptyset,\ 1), \rho|_\alpha := (z), \perp\}$. This is valid MICE' proof of $\Phi|_\alpha$ with the correct model-count.

Lemma 3. *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 $\mathsf{PAC}(0,J)$. According to Lemma 1, this $\mathsf{PAC}(0,J)$ contains a maximum of n partial assignments. For each of these partial assignments $\alpha \in \mathsf{PAC}(0,J)$, follow the procedure described in Theorem 7 (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 2. *From a MICE' proof π of Φ you can efficiently extract cumulator circuits.*

Theorem 8. CLIP^{NP} *simulates* MICE'.

Proof. Corollary 2 demonstrates that one can extract cumulator circuits of polynomial size from any MICE' proof. The proposition $\mathsf{check}(\xi)$ is polynomial in the original propositional formula. Since these circuits are correct, owing to Theorem 7 and Lemma 1, the NP-Oracle in CLIP^{NP} always returns true. □

Theorem 9. *There is a family of propositional tautologies $||\Psi||$ which can be recognized in polynomial time such that* CLIP $+\mathsf{eFrege}$ $+||\Psi||$ *simulates* MICE'.

Proof. This works the same as the proof of Theorem 6 but we substitute MICE' for $\mathsf{KCPS}(\#\mathsf{SAT})$. □

5 Exponential Improvement on Existing #SAT proof systems

In this section, we give short CLIP framework proofs for hard formulas of existing proof systems. In Section 5.1, we give short proofs of XOR-PAIRS in CLIP+eFrege system (Theorem 10). These formulas were previously proven to be hard for MICE′ system [9, Theorem 23]. In Section 5.2, we give short proofs of Symm_n in the CLIP framework (Theorem 11). The function corresponding to these formulas are shown to be hard for KCPS(#SAT) [34, Theorem 10.3.8].

5.1 XOR-PAIRS

Definition 13 (XOR-PAIRS [9])**.** *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 systems MICE and MICE′ [9, Theorem 23]. We will show in Theorem 10 that these formulas are easy in CLIP.

Definition 14. *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 $(\neg a \vee b) \wedge (\neg b \wedge a)$. 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 4. *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 5. *Let γ and δ be vectors of n variables. Recall Definitions 4 and 14. 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 14), 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 6. *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 14), 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 14), 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 7. *Let α, β and γ be vectors of n variables. Recall Definitions 4 and 14. 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 8. *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 9. *Let α, β and γ be vectors of n variables. Recall Definitions 4 and 14. 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 14), 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 10. 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. 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 5 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 5 to make 3 cases.

1. Let $B_Z = P(B_X)$, we can get a short eFrege proof of $\neg L(B_Z, P(B_X))$ (Lemma 5) and $L(A_Z, P(A_X))$ (Lemma 9), 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 get a short eFrege 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 of $\neg L(B_Z, P(B_X))$ (Lemma 5) and $\neg L(A_Z, P(A_X))$ (Lemma 7), 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 8 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 $\xi(2^{|X|} - 1, 2^{|Z|} - 1) = 2^{|X|} - 1 + 1 = 2^{|X|}$. □

5.2 Permutation matrices

Consider the Boolean function f_{Symm}[1] 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][5, Corollary 3.8] and hence it is hard to count models of the CNF representing f_{Symm} (Symm, Definition 15) with the existing static proof system KCPS(#SAT) [14]. We show that these formulas are easy in CLIP (Theorem 11).

Definition 15. *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. 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 11. 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 ith 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 nth 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. □

For a detailed version of Algorithm 2, see Appendix B. We provide Example 4 to illustrate the working of Algorithm 2.

Example 4. *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 every variable is set to 1, in this case the Algorithm 2 should return the total $\#_{models}(\mathsf{Symm}_3) =$*

[1]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 Pseudocode to compute $C_{models}(\mathsf{Symm}_n, \alpha)$

```
Require: $num(\alpha) < 2^{n^2}$, variables are in row-major order (i.e $x_{1,1}, ..., x_{1,n}, x_{2,1}, ..., x_{2,n}, ...$)
  function CUMULATIVE-SYMM-MODELS(total assignment $\alpha$, int $n$)
      partition $\alpha$ row-wise $\rightarrow J_1, ..., J_n$   /*$J_i$ is the $num(\alpha)$ restricted to the $i^{th}$ row)*/
      int pos_rows = $\{1, 2, ..., 2^{n-1}\}$ /* possible rows in any model*/
      int $i = 1$, int $models = 0$
      for row $\in$ pos_rows do /* pos_rows is always in ascending order*/
          if row $< J_i$ then  /*include all models having row as their $i^{th}$ row*/
              $models \leftarrow models + (n - i)!$
          else if row $> J_i$ then break the loop
          else
              pos_rows $\leftarrow$ pos_rows $\setminus$ row  /*remove row $= J_i$ from the possible rows*/
              restart the loop for $J_{i+1}$ with the updated pos_rows array in ascending order.
          end if
      end for
  return $models$
  end function
```

$C_{models}(\mathsf{Symm}_3, 511) = 6$. *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.

Since here we used an NP-oracle the separation is incomplete. To show a separation using CLIP+eFrege, one needs eFrege-refutation of the cumulator's correctness from Theorem 11.

6 Conclusion

We have shown the existence of the CLIP framework for propositional model counting and demonstrated its advantages. Our approach here has been theoretical and no version of CLIP has been implemented.

Future work on applicable checking formats that iterate on the strength we have demonstrated in CLIP should take into account weighted and projected model counting. For future theory, work needs to be done to understand CPOG in terms of proof complexity, and in comparison to the existing systems including CLIP proof systems. In propositional logic, DRAT has been shown to be p-equivalent to eFrege [35], but surprisingly in QBF QRAT has been shown to be conditionally separated from eFrege + $\forall$red[20]. It is difficult to know what result to expect. The theoretical results presented in this paper gesture towards a proof complexity result, which is yet to be fully proved. We can formulate this as a conjecture.

Conjecture 1. CLIP+eFrege *simulates* MICE$'$ *and* KCPS(#SAT).

A similar conjecture was made in [18] for QBF, where a number of proof systems were listed and each conjectured to be simulated by QBF extended Frege, see Conjecture 1 in [18]. Most have since been simulated [21, 19]. The techniques used in these papers may be useful here. One technique [21] is local extraction, in QBF we would be extracting Herbrand functions, in model counting we would be extracting the cumulator circuits. The locality refers to the circuits being attached to a proof (MICE') line, so each line has a local cumulator. Simultaneously we would take a propositional interpretation of the correctness of each line and make sure the cumulator affirms the line in an eFrege proof. The eFrege proof would be inductive until we get to the final line. The other technique [19] involves arguing for correctness when hitting a proof by a restriction. This could be very useful in MICE' since it has closure under restrictions, and restrictions are used to calculate a cumulator circuit. Either way the QBF proofs use large amounts of extension variables which need to be defined correctly and kept track of, so the actual proof of simulation could well be substantial, but likely to be possible.

Acknowledgements

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

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009.

[2] Rehan Abdul Aziz, Geoffrey Chu, Christian J. Muise, and Peter J. Stuckey. #∃SAT: Projected model counting. In Marijn Heule and Sean A. Weaver, editors, *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *Lecture Notes in Computer Science*, pages 121–137. Springer, 2015.

[3] Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pages 154–169, 2014.

[4] Roberto J Bayardo Jr and Joseph Daniel Pehoushek. Counting models using connected components. In *AAAI/IAAI*, pages 157–162, 2000.

[5] 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.

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

[7] Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. Proof complexity of resolution-based QBF calculi. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 76–89. LIPIcs series, 2015.

[8] 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.

[9] 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.

[10] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

[11] Elazar Birnbaum and Eliezer Lozinskii. The good old Davis-Putnam procedure helps counting models. *Journal of Artificial Intelligence Research*, 10, 07 1999.

[12] Randal E Bryant, Wojciech Nawrocki, Jeremy Avigad, and Marijn JH 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.

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

[14] 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.

[15] Florent Capelli, Jean-Marie Lagniez, and Pierre Marquis. Certifying top-down decision-DNNF compilers. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 6244–6253. AAAI Press, 2021.

[16] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. Approximate model counting. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1015–1045. IOS Press, 2021.

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

[18] Leroy Chew. Hardness and optimality in QBF proof systems modulo NP. In *SAT 2021*, pages 98–115, Cham, 2021. Springer.

[19] Leroy Chew. Proof simulation via round-based strategy extraction for QBF. *Electron. Colloquium Comput. Complex.*, TR23-053, 2023.

[20] Leroy Chew and Judith Clymo. How QBF expansion makes strategy extraction hard. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part I*, volume 12166 of *Lecture Notes in Computer Science*, pages 66–82. Springer, 2020.

[21] Leroy Chew and Friedrich Slivovsky. Towards uniform certification in QBF. In *39th International Symposium on Theoretical Aspects of Computer Science*, page 1, 2022.

[22] 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.

[23] Adnan Darwiche. New advances in compiling CNF to decomposable negation normal form. In *Proceedings of the 16th European Conference on Artificial Intelligence*, ECAI'04, page 318–322, NLD, 2004. IOS Press.

[24] Peter M. Fenwick. A new data structure for cumulative frequency tables. *Softw. Pract. Exp.*, 24(3):327–336, 1994.

[25] 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.

[26] 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.

[27] Carla P. Gomes, Jörg Hoffmann, Ashish Sabharwal, and Bart Selman. From sampling to model counting. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 2293–2299, 2007.

[28] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 993–1014. IOS Press, 2021.

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

[30] Marijn Heule, Martina Seidl, and Armin Biere. A unified proof system for QBF preprocessing. In *7th International Joint Conference on Automated Reasoning (IJCAR)*, pages 91–106, 2014.

[31] 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.

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

[33] Mikoláš Janota and Joao Marques-Silva. Expansion-based QBF solving versus Q-resolution. *Theor. Comput. Sci.*, 577:25–42, 2015.

[34] Samuel D. Johnson. Branching programs and binary decision diagrams: theory and applications by ingo wegener society for industrial and applied mathematics, 2000 408 pages. *SIGACT News*, 41(3):36–38, 2010.

[35] 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.

[36] Tuukka Korhonen and Matti Järvisalo. SharpSAT-TD in model counting competitions 2021-2023, 2023.

[37] Lukas Kroc, Ashish Sabharwal, and Bart Selman. Leveraging belief propagation, backtrack search, and statistics for model counting. *Ann. Oper. Res.*, 184(1):209–231, 2011.

[38] 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.

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

[40] John Alan Robinson. Theorem-proving on the computer. *Journal of the ACM*, 10(2), 1963.

[41] Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996.

[42] Tian Sang, Fahiem Bacchus, Paul Beame, Henry A. Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. In *International Conference on Theory and Applications of Satisfiability Testing*, 2004.

[43] Agnes Schleitzer and Olaf Beyersdorff. Classes of Hard Formulas for QBF Resolution. In *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 5:1–5:18, 2022.

[44] Nathan Segerlind. The complexity of propositional proofs. *Bull. Symb. Log.*, 13(4):417–481, 2007.

[45] Marc Thurley. sharpSAT – counting models with advanced component caching and implicit BCP. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006*, pages 424–429, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[46] Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[47] G. S. Tseitin. *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983.

[48] Allen Van Gelder. Contributions to the theory of practical quantified Boolean formula solving. In *Principles and Practice of Constraint Programming*, pages 647–663. Springer, 2012.

[49] 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.

[50] Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified Boolean satisfiability solver. In *ICCAD*, pages 442–449, 2002.

Appendix

A Algorithm from Section 4

We provide a more detailed version of Algorithm 1.

Algorithm 1 Fenwick tree [24] 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*/
          $\alpha$.append(parent)
          dash.append($log(indx -$ parent)) /*records number of variables to forget from the corre-
  sponding total assignment $\alpha$*/
          $indx \leftarrow$ 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 Algorithm from Section 5.2

We provide a more detailed version of Algorithm 2.

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 $\alpha$, 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 $\mathsf{pos\_rows} = \{1, 2, 4, 8, ..., 2^{n-1}\}$ /* possible rows in any model*/
      int $i = 1$, int $models = 0$, int flag $= 0$
      while flag=0 do
          for $\mathsf{row} \in \mathsf{pos\_rows}$ do
              if $\mathsf{row} < J_i$ then /*include all models having row as the $i^{th}$ row in them*/
                  $models = models + (n - i)!$
              else if $\mathsf{row} > J_i$ 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
                  $\mathsf{pos\_rows} = \mathsf{pos\_rows} \setminus \mathsf{row}$
                  break-loop
              end if
              flag= 1
          end for
      end while
  return $models$
  end function
```