Proof Complexity of Propositional Model Counting

Anil Shukla

Indian Institute of Technology Ropar

Complexity Theory Update Meeting
IMSc Chennai
January 23, 2025

Credits

The credit for my work and understanding on propositional model counting –

- **Sravanthi Chede** IIT Ropar, Rupnagar, India.
- **Leroy Chew** TU, Wien, Austria.

Talk Contents

Proof Systems

- A proof system $f : \Delta^* \to \Sigma^*$ for a language $L \in \Sigma^*$ is a polynomial-time computable function such that the range of f is L.
- For $x \in L$, if $f(w) = x$, then w is an f-proof of the fact that $x \in L$. $|w|$ is the length of the proof.
- **Soundness**: For any $x \in \Sigma^*$ and $w \in \Delta^*$, if $f(w) = x$, then $x \in L$.
- **Completeness**: For every $x \in L$, there must exists $w \in \Delta^*$ such that $f(w) = x$.

Proof Systems

- Let f and g are two proof systems for a language L. we say that f **simulates** g if there is a computable function A that transforms the proofs in g to proofs in f with at most a polynomial blow in size.
- If A is polynomial time computable, then we say that f **p-simulates** g.
- f and g are p-equivalent if both can p-simulates each other.
- If f p-simulates g but g does not p-simulate f then we say that f is strictly stronger than g.
- We say that f and g are incomparable if both cannot p-simulates each other.
- Proof systems for the language UNSAT are called propositional poof systems. For example, the Resolution proof system.

Resolution (Res) Proof System [Blake 1937, Davis and Putnam 1960, Robinson 1965]

- Resolution rule: $\frac{(C \vee x) \quad (D \vee \neg x)}{(C \vee D)}$, here C and D are any clauses.
- Let F be an unsatisfiable CNF formula. A Resolution proof π of F is a sequence of clauses

$$D_1, D_2, \ldots, D_k$$

such that the last clause D_k is the empty clause and each D_i obeys one of the following

  - $D_i \in F$
  - D_i is derived from some clauses D_k, D_j, with $j, k < i$ via the resolution rule.

Reverse Unit Propogation [Goldberg and Novikov 2003]

- **Unit propagation (UP)**: Unit propagation satisfies the unit clauses of the CNF formula F by assigning their literal to true. Until you get a fix point or a conflict (x and $\neg x$ both become true for some variable x).
- Given an assignment α, $F|_\alpha$ denotes the CNF formula F' without the clauses of F satisfied by α and without the literals in the clauses of F falsified by α.
- Let F be a CNF formula and C a clause. Let α be the smallest assignment that falsifies C. We say that C is implied by F through UP (denoted $F \vdash_1 C$) if UP on $F|_\alpha$ results in a conflict.
- $F \vdash_1 C$ is known as Reverse Unit Propagation.

Propositional Model Counting

- For a given CNF formula F, the propositional model counting #SAT problem asks to compute the number of satisfying assignments.
- #SAT is one of the hardest known problems in the field of computational complexity.
- In fact, Toda [1991] shows that with a single call to a #SAT oracle, any problem in the polynomial hierarchy can be solved in polynomial time.
- In this talk, we focus on different proof systems for the propositional model counting problem.
- To be precise, we focus on different proof systems for the language $L = \{(F, k) \mid F \text{ has exactly } k \text{ satisfying assignments}\}$

A Naive Proof Systems for #SAT

- A naive proof system to prove that a CNF formula F has exactly k satisfying assignment is to list the k satisfying assignments along with a resolution proof of the CNF formula F', where F' consist of the following clauses:
  - All clauses of F belongs to F'.
  - For each satisfying assignment α of F, there is a clause $C_\alpha \in F'$, where C_α is the clause that has the unique falsifying assignment α.

A Naive Proof Systems for #SAT

- A naive proof system to prove that a CNF formula F has exactly k satisfying assignment is to list the k satisfying assignments along with a resolution proof of the CNF formula F', where F' consist of the following clauses:
  - All clauses of F belongs to F'.
  - For each satisfying assignment α of F, there is a clause $C_\alpha \in F'$, where C_α is the clause that has the unique falsifying assignment α.
- Example: $(x \vee \overline{y}) \wedge (\overline{x} \vee y)$, satisfying assignments are $\{x = 0, y = 0\}$ and $\{x = 1, y = 1\}$.
  $F' := (x \vee \overline{y}) \wedge (\overline{x} \vee y) \wedge (x \vee y) \wedge (\overline{x} \vee \overline{y})$
  Res-proof of F':

$$\dfrac{\dfrac{(x \vee \overline{y}) \qquad (x \vee y)}{(x)} \qquad \dfrac{(\overline{x} \vee y) \qquad (\overline{x} \vee \overline{y})}{(\overline{x})}}{\bot}$$

MICE Proof System

- Inspired from many #SAT solvers, Fitch et al. SAT-2022, designed a proof system MICE (**Model Counting Induction by Claim Extension**) for #SAT. A simplified and equivalent proof system MICE' is designed by Beyersdorff et al.SAT-2023. These systems work with claims.
- **Claims**: A claim is a 3-tuple (F, A, c), where F is a CNF formula, A is a partial assignment over the vars(F), and c is a count.
- We say that a claim is correct if c is equal to the number of satisfying assignments of $F|_A$.

Definition (MICE, Fitch et al.SAT-2022, Beyersdorff et al.SAT-2023)

A MICE proof of a CNF formula F is a sequence of claims $I_1, I_2, \dots, I_k$ that are derived from inference rules Axiom, Composition, Join, and Extension, such that the final claim I_k is a correct claim of the form $(F, \emptyset, c)$.

Inference rules of MICE

- **Axioms**:

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

Inference rules of MICE

- **Axioms**:

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

- **Composition**:

$$\frac{(F, A_1, c_1) \quad \cdots \quad (F, A_n, c_n)}{(F, A, \sum_{i \in [n]} c_i)}$$

C-1 $\text{vars}(A_1) = \cdots = \text{vars}(A_n)$ and $A_i \neq A_j$ for $i \neq j$.
C-2 $A \subseteq A_i$, for all $i \in [n]$.
C-3 There exists a resolution proof of the CNF formula $A \cup F \cup \{\overline{A_i} \mid i \in [n]\}$.
This proof is called the absence of models statement.

Inference rules of MICE

- **Join**:

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

J-1 A_1 and A_2 are consistent.
J-2 $\text{vars}(F_1) \cap \text{vars}(F_2) \subseteq \text{vars}(A_i)$ for $i \in \{1, 2\}$.

Inference rules of MICE

- **Join**:

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

J-1 A_1 and A_2 are consistent.
J-2 $\text{vars}(F_1) \cap \text{vars}(F_2) \subseteq \text{vars}(A_i)$ for $i \in \{1, 2\}$.

- **Extension**:

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

E-1 $F_1 \subseteq F$,
E-2 $A|_{\text{vars}(F_1)} = A_1$,
E-3 A satisfies $F \setminus F_1$.

- Since all the rules are sound, MICE proof system is sound.

Complexity measures of MICE

- **Size of** π: Let π be a MICE proof of F. Then $s(\pi)$ denotes the size of π which is the total number of claims plus the number of clauses in the resolution proofs in the absence of models statements.
- $c(\pi)$: Another complexity measure is the number of claims in the MICE proof π. This is denoted by $c(\pi)$.

MICE Proof system is Complete

Satisfying Assumption Rule (SA): If A satisfies F, we are allowed to derive the following: $\overline{(F, A, 2^{|\text{vars}(F) \setminus \text{vars}(A)|})}$

Theorem (Fitch et al.SAT-2022, Beyersdorff et al.SAT-2023)

MICE is complete

Proof.

Let F be a CNF formula, and Mod(F) denotes the set of all satisfying assignments of F.
For every assignment $\alpha \in \text{Mod}(F)$, derive $I_\alpha = (F, \alpha, 1)$ via SA.
For all these models, there must be an absence of model statement.
Derive $(F, \emptyset, |\text{Mod}(F)|)$ using the composition rule. □

Corollary (Beyersdorff et al.SAT-2023)

Every CNF formula F has a MICE proof π with $c(\pi) = |Mod(F)| + 2$.

Lower Bounds for MICE

Theorem (Beyersdorff, Hoffmann, Spachmann SAT-2023)

MICE is p-equivalent to Res for unsatisfiable formulas.

- Pigeonhole formulas PHP_n are hard for Res [Haken 1985].

Corollary (Beyersdorff, Hoffmann, Spachmann SAT-2023)

Any MICE proof π of PHP_n has size $s(\pi) = 2^{\Omega(n)}$.

- These lower bounds are not so interesting. As these lower bounds are implied from Res lower bounds.
- PHP_n has a MICE proof π of just one step, i.e., $c(\pi) = 1$.
- Interesting problem: Show lower bounds on the number of claims in the MICE proof.

MICE Lower Bounds on the number of Inference Steps

- Recall that any CNF formula F has a MICE proof π such that $c(\pi) \leq |\mathrm{Mod}(F)| + 2$
- In order to prove the number of claims lower bounds, we must pick CNFs with exponentially many satisfiable assignments.

Definition (XOR PAIRS$_n$)

The formula XOR-PAIRS$_n$ consists of the following clauses:

$$C_{ij}^1 = (x_i \vee x_j \vee \overline{z_{ij}}), \qquad C_{ij}^2 = (\overline{x_i} \vee x_j \vee z_{ij})$$
$$C_{ij}^3 = (x_i \vee \overline{x_j} \vee z_{ij}), \qquad C_{ij}^4 = (\overline{x_i} \vee \overline{x_j} \vee \overline{z_{ij}})$$

for $i, j \in [n]$.

- XOR-PAIRS$_n$ satisfies exactly if $z_{ij} = x_i \oplus x_j$. XOR-PAIRS$_n$ has 2^n models.

MICE Lower Bounds on the number of Inference Steps

Theorem

Any MICE proof π of XOR-PAIRS$_n$ requires claims $c(\pi) = 2^{\Omega(n)}$.

Proof Idea:

- The final claim of a MICE proof must have a large count (i.e., 2^n).
- MICE proof always begin with a small count (i.e., 1).
- In order to reach a large count from a small count with minimum number of steps, a MICE proof must use the Extension or Join steps. Since in these steps, the count gets mutiplied.
- For XOR-PAIRS$_n$, one factor of any such multiplication is always a 1.
- Thus, the only way to increase the count is through the composition rule.
- To reach the count 2^n from 1, exponential number of summands (i.e., composition rules) are required.

Knowledge Compilation

- Next two #SAT proof systems KCPS(#SAT) and CPOG use concepts from knowledge compilation.
- Knowledge compilation has emerged as a new direction of research for dealing with the computational intractable problems like propositional model counting.
- This technique compiles off-line a propositional theory (like CNF formulas) into a target language (like some well studied structures say DNNF).
- The target language (like DNNF) is then used on-line to answer a large number of queries in polynomial time.
- Let us discuss some important target languages used.

Knowledge Compilation: Circuits

- A circuit is a directed acyclic graph with labelled nodes that are called gates. There is a unique gate with in-degree 0, called the root.
- Gate with out-degree 0 are called leaves and are labelled with literals or constant 0 or 1.
- Every inner gate is an AND-, OR-, or NOT-gate and is labelled with the corresponding Boolean function.
- Let D be a circuit. For gates of D, we use uppercase letters such as N.
- $\mathrm{vars}(D)$ denotes the set of all variables that occur in the leaves of D.
- $\mathcal{E}(D)$ denotes a proper encoding of D, where we use a new variable V_N for every gate N.
- $D(N)$ denotes the subcircuit of D with root and N consisting of all descendants of N in D.

Knowledge Compilation: NNF, DNNF, d-DNNF

- A circuit is in **negation normal form** (NNF) if it does not contain NOT-gates.
- An AND-gate with children N_1 and N_2 is called **decomposable**, if $\text{vars}(D(N_1)) \cap \text{vars}(D(N_2)) = \emptyset$.
- An OR-gate with children N_1 and N_2 is called **deterministic** if there is no assignment that satisfies both $D(N_1)$ and $D(N_2)$.
- A **DNNF** (decomposable negation normal form, by Adnan Darwiche, IJCAI-1999), is an NNF where every AND-gate is decomposable.
- A **d-DNNF** (deterministic decomposable NNF, by Adnan Darwiche, JANCL-2001) is a DNNF where every OR-gate is deterministic.

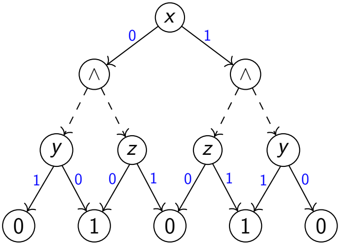
Knowledge Compilation: decision-DNNF

- It is non-trivial to check if all OR-gates are deterministic.
- Decision-DNNF is a restricted version of d-DNNF.
- In a decision-DNNF, any OR-gate has the form $N = (N_1 \text{ or } N_2)$ with $N_1 = (x \text{ and } N_3)$ and $N_2 = (\overline{x} \text{ and } N_4)$ for any variable x.
- Any such OR-gates are deterministic.
- Thus decision-DNNF uses decision gates instead of OR-gates.
- We can assume that the leaves of a Decision-DNNF contain only constants 0 or 1

decision-DNNF

decision-DNNF D over variable set X:

- leaves are either 0 or 1
- decision nodes labeled with x
  - outgoing edges labeled with 0,1
  - x is not repeated in any root-to-leaf path
  - We say that x is tested in D
- decomposable $\wedge$ nodes
- vars(D) = set of variables tested in D

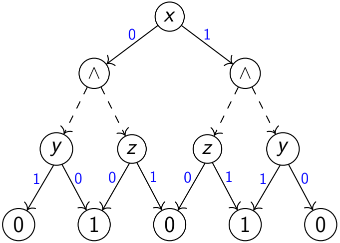


decision-DNNF for $x = y = z$

decision-DNNF

decision-DNNF D **over variable set** X:

- Let $\alpha \in \{0,1\}^X$. A source-sink path P in D is compatible with α if and only if when x is tested on P, the outgoing edge labeled with $\alpha(x)$ is in P.
- We say that α satisfies D, if only 1-gates are reached by paths compatible with α.
- Example: Consider an assignment $\alpha : x = 0, y = 0, z = 1$. α does not satisfy D. Since, it is reaching a 0-gate.

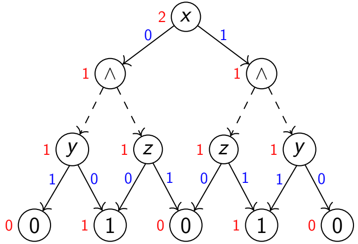


decision-DNNF D for $x = y = z$

Propositional Model Count is easy for decision-DNNF

Theorem (Adnan Darwiche 2001)

Given a decision-DNNF D for a CNF formula F such that $D \equiv F$, it is easy to compute the $|Mod(F)|$.



decision-DNNF D for $x = y = z$

- count models in bottom-up fashion
- assign $0 \rightarrow$ 0-sinks, $1 \rightarrow$ 1-sinks.
- at an $\wedge$-gate: mutiply the model count of both children
- at a decision-gate: let two child nodes be N_1, N_2. Then, model-count $=$ $(2^{|vars(N_1) \setminus vars(N_2)|} \times$ count of $N_2) + (2^{|vars(N_2) \setminus vars(N_1)|} \times$ count of $N_1)$

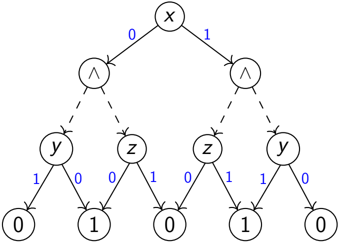
Knowledge Compilation based Proof System for #SAT (KCPS(#SAT))

A KCPS(#SAT) proof of a CNF F provides a decision-DNNF D such that $D \equiv F$.

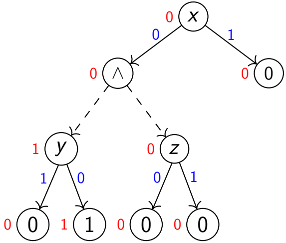
$F := (x \vee \overline{y}) \wedge (\overline{x} \vee y) \wedge (x \vee \overline{z}) \wedge (\overline{x} \vee z)$

Say, $C = (x \vee \overline{z})$. Does decision-DNNF $D \implies C$?

Easy, check if $D \wedge \overline{C}$ is UNSAT.



decision-DNNF D for $x = y = z$



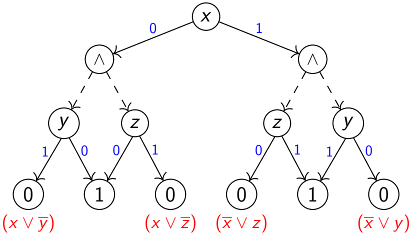
decision-DNNF $D|_{\overline{C}}$

(conditioning D with $\{x = 0, z = 1\}$)

$D \implies F$ ✓

KCPS(#SAT) contd.

$F := (x \vee \overline{y}) \wedge (\overline{x} \vee y) \wedge (x \vee \overline{z}) \wedge (\overline{x} \vee z)$



certified decision-DNNF D for F

Given a decision-DNNF D and a CNF F, it is coNP-complete to check whether $F \implies D$ [Capelli, SAT 2019]

Certified decision-DNNF D over X:

- every 0-sink M has a label of a clause $C_M \in F$ s.t.
  - for every $\alpha \in \{0,1\}^X$, s.t there is a path from source to a 0-sink M compatible with α, we have α falifies C_M.

Let $F(D) :=$ set of all 0-sink labels i.e. $F \implies F(D)$.

$F(D) \implies D$: every α falsifying D ends up at a 0-sink
hence α also falsifies the label $C \in F(D)$ at that sink.

$F \implies D$ ✓

KCPS(#SAT) contd.

Definition (KCPS(#SAT), Capelli, SAT-2019)

Given a CNF F, a certificate that F has exactly k satisfying assignment is a correct certified decision-DNNF D such that:

- every clause of $F(D)$ are clauses of F,
- D computes F and has k satisfying assignments.

For a proof system, the proof must be polynomial time verifiable. The verification process is simple:

- Check that D is correct. That is, check if all labeled clauses at the 0-sinks are correct. This is easy shown by Capelli, SAT-2019.
- Check that $D \equiv F$. Easy
- Check if k satisfying assignment of D. Easy!

Lower bounds for KCPS(#SAT)

- Many lower bounds on the size of decision-DNNFs representing CNFs already known [Paul Beame et al.UAI, 2013, Simone Bova et al., IJCAI,2016]
- For all such CNF formulas, we have KCPS(#SAT) lower bounds.

Theorem (Olaf Beyersdorff et al., SAT-2024)

For unsatisfiable formulas, KCPS(#SAT) *and regular resolution are p-equivalent.*

- All unsatisfiable CNFs which are hard for regular resolution are also hard for KCPS(#SAT)

CPOG: Certified Partitioned Operation Graph

- Another proof system CPOG for #SAT is designed by Randal E. Bryant et al., SAT-2023.
- CPOG is not restricted to the weak certified decision-DNNF, but uses more flexible circuit class POG (Partitioned Operation Graph).
- Model counting is efficient for POG.

Definition (Partitioned Operation Graph)

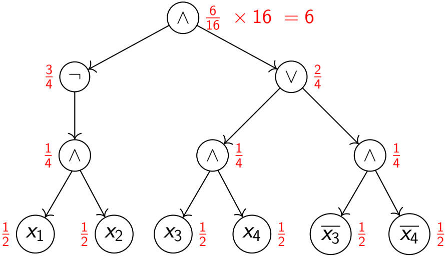
A POG is a d-DNNF with NOT-gates.

- Every AND-gate is decomposable, OR-gate is deterministic, and NOT-gates are allowed in POG.
- Alternatively, a d-DNNF can be viewed as a POG with negation applied only to variables.

Example: Partitioned Operation Graph

$F := (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_4})$

An assignment α satisfies the POG if evaluating it on α evaluates to True. For instance, $\{x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1\}$ is a model for P.



POG P for F

Model counting is easy in POG:

- count in bottom-up fashion
- assign $0 \to$ 0-sinks, $1 \to$ 1-sinks, $\frac{1}{2} \to$ literals
- at an $\wedge$-gate: multiply the model count of all children
- at a $\neg$-gate: model count $= 1-$ count at the child node
- at an $\vee$-gate: add the model count of both children
- at root: Final model count $=$ multiply count with $2^{|vars(F)|}$.

CPOG: Certified Partitioned Operation Graph

- Since a proof system must be polynomial time verifiable, any proof system which uses POG to certify #SAT of a CNF formula F must include the following information as well:
  - Encoding $\mathcal{E}(P)$ of the POG P,
  - A proof of the fact that $F \implies P$,
  - A proof of the fact that $P \implies F$,
  - A proof that all the OR-gates used in P are indeed deterministic.
- A POG for a CNF formula F including all the above information is a CPOG proof.

The CPOG representation and the Proof System

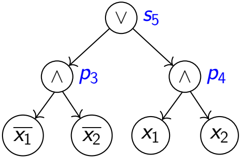
A CPOG proof must contains the following:

- POG representation and clausal encoding of POG $\mathcal{E}(P)$.
- For all OR-gates explicit proof with hints of the fact that they are deterministic.
- For $F \implies P$: The proof contains explicit clause addition steps. A clause can only be added if it is logically implied by the existing clauses. A sequence of clause identifiers must be listed as a hint providing a RUP verification of the implication.
- For $P \implies F$: The proof contains explicit clause deletion steps. A clause can only be deleted if it is logically implied by the remaining clauses. A sequence of clause identifiers must be listed as a hint providing a RUP verification of the implication.

CPOG example (Bryant et al., SAT 2023 slides)

$F := (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$

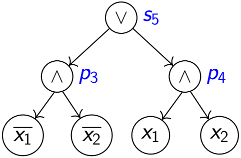
ID	Literals	Explanation
1	1 −2	Input
2	−1 2	Input



CPOG example (Bryant et al., SAT 2023 slides)

$F := (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$

ID	Literals	Explanation
1	1 −2	Input
2	−1 2	Input



POG Declaration:

Type	Literals	Hint
p	3 −1 −2	-

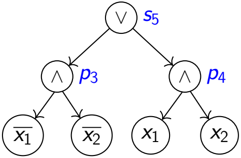
CPOG Proof:

ID	Literals	Explanation
3	3 1 2	p_3
4	−3 −1	
5	−3 −2	

CPOG example (Bryant et al., SAT 2023 slides)

$F := (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$

ID	Literals	Explanation
1	1 −2	Input
2	−1 2	Input



POG Declaration:

Type	Literals	Hint
p	3 −1 −2	-
p	4 1 2	-

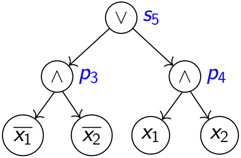
CPOG Proof:

ID	Literals	Explanation
3	3 1 2	p_3
4	−3 −1	
5	−3 −2	
6	4 −1 −2	p_4
7	−4 1	
8	−4 2	

CPOG example (Bryant et al., SAT 2023 slides)

$F := (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$

ID	Literals	Explanation
1	1 −2	Input
2	−1 2	Input



POG Declaration:

Type	Literals	Hint
p	3 −1 −2	-
p	4 1 2	-
s	5 3 4	4 7 $\rightarrow (\overline{p_3} \vee \overline{p_4})$

CPOG Proof:

ID	Literals	Explanation
3	3 1 2	p_3
4	−3 −1	
5	−3 −2	
6	4 −1 −2	p_4
7	−4 1	
8	−4 2	
9	−5 3 4	s_5, 4 7
10	5 −3	(Input clauses not allowed
11	5 −4	in RUP proof of ∨-gates)

$\rightarrow$ RUP proof of $\overline{p_3} \vee \overline{p_4}$

$\Longrightarrow$ p_3, p_4 have disjoint models

CPOG example (Bryant et al., SAT 2023 slides)

$F := (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$

ID	Literals	Explanation
1	1 −2	Input
2	−1 2	Input

RUP Additions:

Type	Literals	Hint
a	−2 5	11 1 6
	↓	↓ ↓ ↓
	2, −5	−4 1 −2

CPOG Proof:

ID	Literals	Explanation
3	3 1 2	p_3
4	−3 −1	
5	−3 −2	
6	4 −1 −2	p_4
7	−4 1	
8	−4 2	
9	−5 3 4	s_5, 4 7
10	5 −3	
11	5 −4	
12	−2 5	11 1 6

CPOG example (Bryant et al., SAT 2023 slides)

$F := (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$

ID	Literals	Explanation
1	1 −2	Input
2	−1 2	Input

RUP Additions:

Type	Literals	Hint
a	−2 5	11 1 6
a	5	10 12 2 3
	↓	↓ ↓ ↓ ↓
	−5	−3 −2 −1 3

CPOG Proof:

ID	Literals	Explanation
3	3 1 2	p_3
4	−3 −1	
5	−3 −2	
6	4 −1 −2	p_4
7	−4 1	
8	−4 2	
9	−5 3 4	s_5, 4 7
10	5 −3	
11	5 −4	
12	−2 5	11 1 6
13	5	10 12 2 3

CPOG example (Bryant et al., SAT 2023 slides)

$F := (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$

ID	Literals	Explanation
1	1 −2	Input
2	−1 2	Input

RUP deletions:

Type	Clause	Hint
d	12	11 1 6

CPOG Proof:

ID	Literals	Explanation
3	3 1 2	p_3
4	−3 −1	
5	−3 −2	
6	4 −1 −2	p_4
7	−4 1	
8	−4 2	
9	−5 3 4	s_5, 4 7
10	5 −3	
11	5 −4	
12	−2 5	11 1 6
13	5	10 12 2 3
d	12	11 1 6

CPOG example (Bryant et al., SAT 2023 slides)

$F := (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$

ID	Literals	Explanation
1	1 −2	Input
2	−1 2	Input

RUP deletions:

Type	Clause	Hint
d	12	11 1 6
d	1	13 5 7 9
	↓	↓ ↓ ↓ ↓
	−1, 2	5 −3 −4 4

CPOG Proof:

ID	Literals	Explanation
3	3 1 2	p_3
4	−3 −1	
5	−3 −2	
6	4 −1 −2	p_4
7	−4 1	
8	−4 2	
9	−5 3 4	s_5, 4 7
10	5 −3	
11	5 −4	
12	−2 5	11 1 6
13	5	10 12 2 3
d	12	11 1 6
d	1	13 5 7 9

CPOG example (Bryant et al., SAT 2023 slides)

$F := (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$

ID	Literals	Explanation
1	1 −2	Input
2	−1 2	Input

RUP deletions:

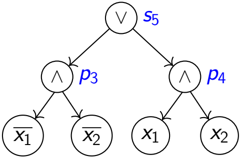
Type	Clause	Hint
d	12	11 1 6
d	1	13 5 7 9
d	2	13 4 8 9
	↓	↓ ↓ ↓ ↓
	1, −2	5 −3 −4 4

CPOG Proof:

ID	Literals	Explanation
3	3 1 2	p_3
4	−3 −1	
5	−3 −2	
6	4 −1 −2	p_4
7	−4 1	
8	−4 2	
9	−5 3 4	s_5, 4 7
10	5 −3	
11	5 −4	
12	−2 5	11 1 6
13	5	10 12 2 3
d	12	11 1 6
d	1	13 5 7 9
d	2	13 4 8 9

CPOG example (Bryant et al., SAT 2023 slides)

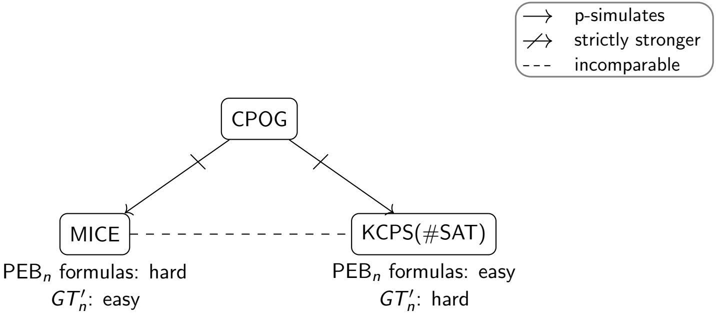
$$F := (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$$



CPOG Proof:

ID	Literals	Explanation
3	3 1 2	p_3
4	−3 −1	
5	−3 −2	
6	4 −1 −2	p_4
7	−4 1	
8	−4 2	
9	−5 3 4	s_5, 4 7
10	5 −3	
11	5 −4	
12	−2 5	11 1 6
13	5	10 12 2 3
d	12	11 1 6
d	1	13 5 7 9
d	2	13 4 8 9

Relationship among #SAT proof systems



- There exists a family of unsatisfiable formulas GT'_n based on the ordering principle which are easy for general Res but are hard for regular Res. [Alekhnovich et al., TOC-2007].
- PEB$_n$ formulas on pyramidal graphs are CNF formulas which are shown to be hard for MICE but easy for KCPS(#SAT) [Beyersdorff et al., SAT-2024].

KCPS and MICE are incomparable

Theorem (Beyersdorff et al.,SAT2024)

KCPS(#SAT) *cannot p-simulate* MICE.

Proof.

- MICE is p-equivalent to Res.
- KCPS(#SAT) is p-equivalent to regular resolution.
- There exists family of CNF formulas which are easy for Res but are hard for regular Res [Alekhnovich et al., TOC-2007].
- Such formulas are easy for MICE but hard for KCPS(#SAT).

□

KCPS and MICE are incomparable

Theorem (Beyersdorff et al., SAT-2024)

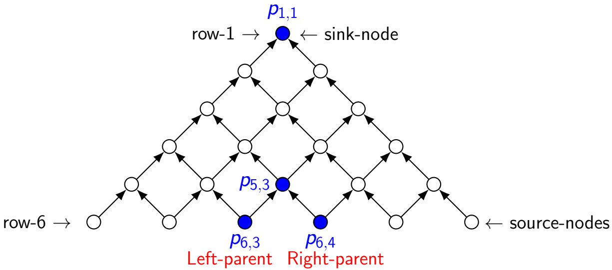
MICE *cannot p-simulate* KCPS(#SAT).

Proof Idea: There exists a family of CNF formulas PEB_n such that it has small certified decision-DNNF D with $D \equiv \text{PEB}_n$ but any MICE proof of PEB_n has size $2^{\Omega(n)}$.

- The CNF formula PEB_n encodes a pebbling game on pyramidal graphs.
- Let us next present the formula PEB_n and an easy KCPS(#SAT) proof for the same.

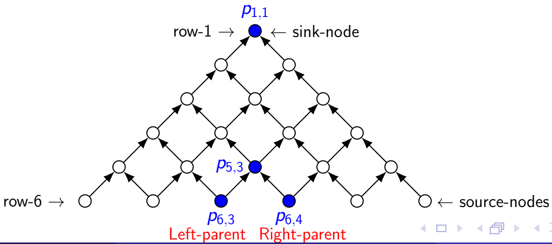
Pyramidal Graph G_n

- PEB_n formulas encode a pebbling game on pyramidal graphs G_n.
- The pyramidal graph G_n has n rows, numbered from 1 to n.
- The row i has i nodes. So, G_n has total $m = \sum_{i=1}^{n} i = n(n+1)/2$ nodes.
- We label each node with $P_{i,j}$, where i corresponds to the row, and j to the column. Clearly, for each node $P_{i,j}$, we have $1 \leq j \leq i \leq n$.
- For each $i < n$, there are edges from $P_{i+1,j}$ and $P_{i+1,j+1}$ to $P_{i,j}$.



PEB_n Formulas

- Before presenting the PEB_n formulas, we briefly discuss the intuition.
- For each node $P_{i,j}$, there are two variables $w_{i,j}$ and $b_{i,j}$.
- $w_{i,j}$ denotes that a white pebble is placed on node $P_{i,j}$.
- $b_{i,j}$ denotes that a black pebble is placed on node $P_{i,j}$.
- PEB_n requires that each source node must contain a pebble (either white or black).
- No node can simultaneously contain a black and a white pebble.
- Every other node needs to contain a pebble if and only if both its parent nodes contain a pebble.



PEB_n Formulas

Definition (PEB_n, Beyersdorff et al.,SAT-2024)

Let n be an integer. The formula PEB_n has variables $w_{i,j}$ and $b_{i,j}$ for every $i,j \in [n]$ with $j \leq i$. The PEB_n is a CNF defined as follows:

– For every $i,j \in [n-1], j \leq i$ the formula requires that

$(w_{i,j} \vee b_{i,j}) \leftrightarrow \Big((w_{i+1,j} \vee b_{i+1,j}) \wedge (w_{i+1,j+1} \vee b_{i+1,j+1})\Big)$ Expressed as:

$$C^1_{i,j} = \overline{w_{i+1,j}} \vee \overline{w_{i+1,j+1}} \vee w_{i,j} \vee b_{i,j} \qquad C^2_{i,j} = \overline{w_{i+1,j}} \vee \overline{b_{i+1,j+1}} \vee w_{i,j} \vee b_{i,j}$$

$$C^3_{i,j} = \overline{b_{i+1,j}} \vee \overline{w_{i+1,j+1}} \vee w_{i,j} \vee b_{i,j} \qquad C^4_{i,j} = \overline{b_{i+1,j}} \vee \overline{b_{i+1,j+1}} \vee w_{i,j} \vee b_{i,j}$$

$$C^5_{i,j} = w_{i+1,j} \vee b_{i+1,j} \vee \overline{w_{i,j}} \qquad C^6_{i,j} = w_{i+1,j} \vee b_{i+1,j} \vee \overline{b_{i,j}}$$

$$C^7_{i,j} = w_{i+1,j+1} \vee b_{i+1,j+1} \vee \overline{w_{i,j}} \qquad C^8_{i,j} = w_{i+1,j+1} \vee b_{i+1,j+1} \vee \overline{b_{i,j}}$$

– For every $i,j \in [n], j \leq i$, there is a clause $C^9_{i,j} = \overline{b_{i,j}} \vee \overline{w_{i,j}}$.

– For every $j \in [n]$, there is a clause $C^{10}_{n,j} = w_{n,j} \vee b_{n,j}$.

PEB$_n$:KCPS(#SAT) proof [Beyersdorff et al.,SAT-2024]

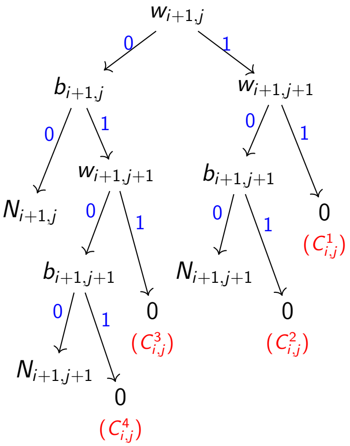
PEB_n:KCPS(#SAT) proof [Beyersdorff et al.,SAT-2024]

$N_{i,j}$: (assuming no pebble at $P_{i,j}$, evaluating bottom-up)

$N_{n,j} \implies 0 \quad (C^{10}_{n,j})$

PEB$_n$:KCPS(#SAT) proof [Beyersdorff et al.,SAT-2024]

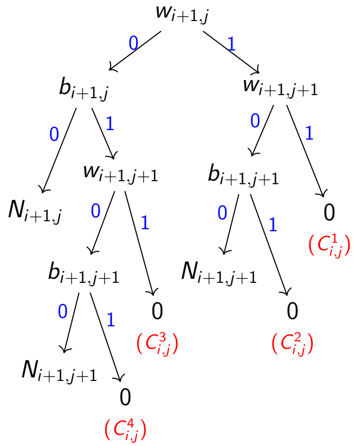
$N_{i,j}$: (assuming no pebble at $P_{i,j}$, evaluating bottom-up)



$N_{n,j} \implies 0 \quad (C^{10}_{n,j})$

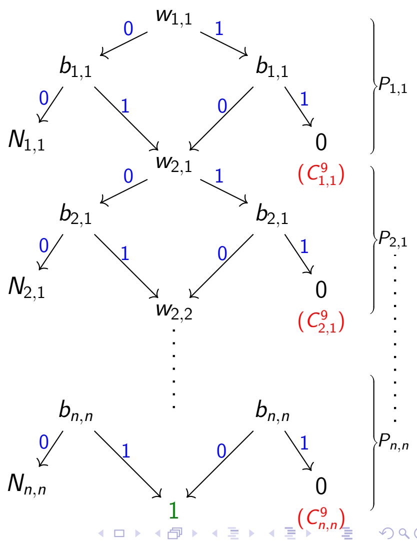
PEB_n:KCPS(#SAT) proof [Beyersdorff et al.,SAT-2024]

$N_{i,j}$: (assuming no pebble at $P_{i,j}$, evaluating bottom-up)

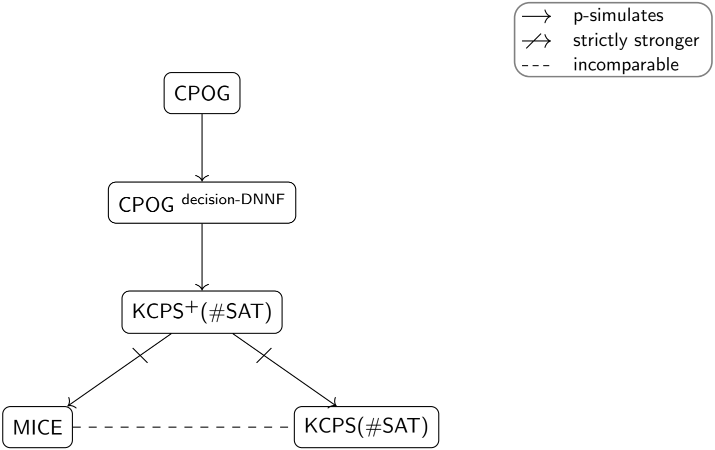


$N_{n,j} \implies 0 \quad (C^{10}_{n,j})$

certified decision-DNNF for PEB_n:



CPOG p-simulates KCPS(#SAT) and MICE



$\mathrm{KCPS}^+(\#\mathrm{SAT})$

- Recall the following definition:

Definition (S-certified Decision DNNF)

Let S be a set of clauses. A decision-DNNF D is called S-certified if every 0-gate N is labelled by a certificate $C \in S$. A clause is a certificate for N if all assignments that reach N falsify C.

Definition ($\mathrm{KCPS}^+(\#\mathrm{SAT})$, Florent Capelli, SAT-2019)

A $\mathrm{KCPS}^+(\#\mathrm{SAT})$ proof of a CNF F is a pair (σ, D) where,

- σ is a Res derivation starting from the clauses in F and
- D is a σ-certified decision DNNF (i.e., all clauses labelling the 0-gates in D are derived in σ) such that $D \equiv F$.

$\text{KCPS}^+(\#\text{SAT})$

Theorem (Beyersdorff et al., SAT-2024)

$\text{KCPS}^+(\#\text{SAT})$ p-simulate KCPS(#SAT).

Proof.

Every KCPS(#SAT) proof D of a CNF formula F can be written as a $\text{KCPS}^+(\#\text{SAT})$ proof (σ, D), where σ contains all clauses from F. □

Theorem (Beyersdorff et al., SAT-2024)

$\text{KCPS}^+(\#\text{SAT})$ p-simulates MICE.

The extraction of the decision-DNNF from a MICE proof was already known. This proof shows how to extract a certified decision-DNNF.

CPOG$^{\text{decision-DNNF}}$

The CPOG$^{\text{decision-DNNF}}$ proof system uses decision-DNNF instead of a POG in the CPOG framework. To be precise,

Definition (CPOG$^{\text{decision-DNNF}}$, Beyersdorff et al., SAT-2024)

A CPOG$^{\text{decision-DNNF}}$ proof of a CNF formula F is a pair $(\mathcal{E}(D), \rho)$ where

- D is a decision-DNNF and $\mathcal{E}(D)$ is a clausal encoding of D such that $D \equiv F$,
- ρ is a proof of $F \implies \mathcal{E}(D)$.

Since decision-DNNF uses decision gates instead of OR-gates, the corresponding proof of CPOG is not needed.
Also, verifying $D \implies F$ is easy.

$\mathrm{CPOG}^{\text{decision-DNNF}}$

Theorem (Beyersdorff et al., SAT-2024)

$\mathrm{CPOG}^{\text{decision-DNNF}}$ p-simulate $\mathrm{KCPS}^{+}(\#\mathrm{SAT})$.

Proof idea: For a CNF formula F, we are given a $\mathrm{KCPS}^{+}(\#\mathrm{SAT})$ proof (σ, D). For the $\mathrm{CPOG}^{\text{decision-DNNF}}$ proof of F, just keep the same decision-DNNF D. Also, using σ it is possible to derive a proof ρ of $F \implies D$.

Theorem (Beyersdorff et al., SAT-2024)

CPOG *p-simulate $\mathrm{CPOG}^{\text{decision-DNNF}}$.*

Proof idea: A decision-DNNF is also a POG.

Conclusion and Open Problems

- In this talk, we discussed about three proof systems for #SAT along with their relationships: MICE, KCPS(#SAT), and CPOG.

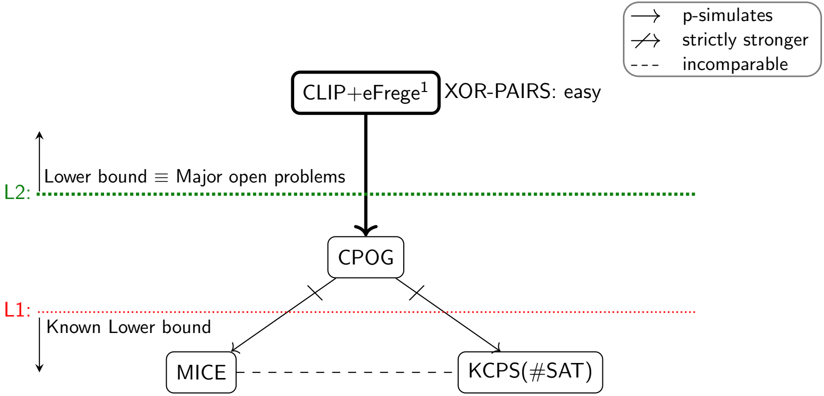
Open Problems:

– Proving lower bounds for POG.

– There exists CNFs with small decision-DNNFs, but requires large certified decision-DNNFs [Beyersdorff et al., SAT-2024]. Similarly, does their exists CNF formulas with small POG, but large CPOG proofs.

– We discussed that XOR-PAIRS formulas are hard for the MICE system. It is open whether XOR-PAIRS formulas are easy or hard for CPOG.

Current #SAT proof complexity



[1]Sravanthi Chede, Leroy Chew, and Anil Shukla. Circuits, Proofs and Propositional Model Counting. FSTTCS 2024.

Thank you.