Proof Systems That Tightly Characterise Model Counting Algorithms

Olaf Beyersdorff, Tim Hoffmann, Kaspar Kasche

Friedrich Schiller University Jena, Germany
olaf.beyersdorff@uni-jena.de, hoffmann.t@uni-jena.de, kaspar.kasche@uni-jena.de

Abstract

Several proof systems for model counting have been introduced in recent years, mainly in an attempt to model #SAT solving and to allow proof logging of solvers. We reexamine these different approaches and show that: (i) with moderate adaptations, the conceptually quite different proof models of the dynamic system MICE and the static system of annotated Decision-DNNFs are equivalent and (ii) they tightly characterise state-of-the-art #SAT solving. Thus, these proof systems provide a precise and robust proof-theoretic underpinning of current model counting. We also propose new strengthenings of these proof systems that might lead to stronger model counters.

1 Introduction

Given a propositional formula φ, the model counting problem #SAT asks how many satisfying assignments exist for φ. This generalises the famous SAT problem, both in terms of computational complexity – where we see a jump from NP (Cook 1971) to #P, encompassing almost all of the polynomial hierarchy (Toda 1991) – and in terms of applications (Bacchus, Dalmao, and Pitassi 2003; Latour et al. 2017; Shi et al. 2020; Baluta et al. 2021; Zhai et al. 2020; Dueñas-Osorio et al. 2017). For SAT solving, the main breakthrough happened in the late 1990s with the introduction of conflict-driven clause learning (CDCL) (Marques Silva, Lynce, and Malik 2021). Since then, SAT solving has matured into a very successful and in terms of applications extremely versatile technology (Biere et al. 2021). Model counting in comparison is in an earlier stage than SAT. There are two different problem settings: exact counting, considered here, and approximate counting (Chakraborty, Meel, and Vardi 2021). These admit fundamentally different algorithmic approaches. Several state-of-the-art model counters, including D4 (Lagniez and Marquis 2017), DSHARP (Muise et al. 2012) and sharpSAT (Thurley 2006), successfully count on a large variety of formulas and regularly participate in the annual model counting competition (Fichte, Hecher, and Hamiti 2021; Fichte and Hecher 2025).

One of the standard approaches as described by Capelli, Lagniez, and Marquis (2021) is to use the classical DPLL branching algorithm for SAT (Davis and Putnam 1960; Davis, Logemann, and Loveland 1962) and optimise it for counting via formula caching and decomposition into variable-disjoint subformulas.

From a theoretical point of view, the enormous success of SAT and #SAT solvers poses an intricate problem: why are these algorithms so successful and where are their limitations? Proof complexity provides one of the primary approaches towards this problem. For SAT, seminal work of Pipatsrisawat and Darwiche (2011) has tightly characterised CDCL (with freely choosable heuristics) by propositional resolution. This implies that lower bounds for resolution proof size translate to lower bounds for the running time of CDCL. Here we pursue a similar characterisation for the classical DPLL-based #SAT approach that we mentioned above.

A second connection comes through proof logging whereby SAT runs for unsatisfiable formulas are efficiently mapped to resolution or in fact optimised, stronger proof systems (Wetzler, Heule, and Hunt Jr. 2014). Recently, this has also been studied for #SAT and a number of proof systems for proof logging have been proposed, including MICE (Fichte, Hecher, and Roland 2022), CPOG (Bryant et al. 2023), and a number of systems using annotated Decision-DNNFs (Capelli 2019; Capelli, Lagniez, and Marquis 2021).

For model counting, the connection to proof complexity was preceded by the discovery that #SAT solvers are intricately linked to knowledge compilation and in fact, Decision-DNNFs – the standard circuit format in knowledge compilation (Huang and Darwiche 2007) – can be extracted from almost all model counters (Capelli 2017). However, the DNNFs themselves are not sufficient for certification as it is hard to check whether a DNNF corresponds to the input CNF. This was realised by Capelli (2019) who introduced the first #SAT proof system kcps using annotated Decision-DNNFs, and subsequent proof systems (Capelli, Lagniez, and Marquis 2021; Bryant et al. 2023) follow this approach. As these systems all use annotated circuits, they are static proof systems. The only exception is the line-based proof system MICE (Fichte, Hecher, and Roland 2022; Beyersdorff, Hoffmann, and Spachmann 2024), but this system also allows for efficient extraction of Decision-DNNFs (Beyersdorff, Hoffmann, and Spachmann 2024). The rela-

tive strength of these different calculi was recently determined by Beyersdorff et al. (2024).

Our contributions. We summarise our main findings.

A. Characterising solver running time by proof size. Our first contribution is a tight characterisation of DPLL-style model counting in the framework of Capelli, Lagniez, and Marquis (2021) (called KC^{Alg}_{Syn} here) by a new proof system that we call KC^{PS}_{Syn}. This builds on prior work of Capelli et al. (2021), who constructed certifiable Decision-DNNFs from runs of the solvers. While it is not clear if these Decision-DNNFs characterise runtime of solvers, we show that this holds for our new proof system KC^{PS}_{Syn}, a slightly adapted version of certifiable Decision-DNNFs. This provides an analogue of the characterisation of CDCL by resolution (Pipatsrisawat and Darwiche 2011).

B. Equivalence to augmented MICE. We further show that our new proof system KC^{PS}_{Syn} is quite robust and natural as it is equivalent to a version of MICE augmented by one natural rule for caching (or more formally, syntactic formula substitution). While the original system MICE is exponentially weaker than KC^{PS}_{Syn} (Theorem 4.1), MICE with this rule precisely captures KC^{PS}_{Syn} and therefore also the solving approach KC^{Alg}_{Syn} (Theorem 4.2). This is quite surprising as these systems use very different approaches.

C. Investigating stronger caching. Modern #SAT solvers apply formula caching in a syntactical setting. We investigate semantic caching – checking formulas for semantic equivalence – and show that the corresponding proof system KC^{PS}_{Sem} characterises solving with semantic caching (Theorem 5.2) and is also equivalent to MICE with a semantic substitution rule (Theorem 5.3). Further, we show that KC^{PS}_{Sem} proof size (neglecting the size of propositional annotations) of a CNF φ is characterised by the Decision-DNNF representation size of φ (cf. Proposition 5.6). This indicates that semantic caching is quite powerful and may be worth investigating in a practical setting.

Organisation. In Section 2 we review notions from proof complexity and knowledge representation. Sections 3, 4, and 5 contain our results described under A, B, and C above, respectively. We conclude in Section 6 with an overview of the emerging landscape of #SAT proof systems (Figures 5 and 6) and outline questions for future research.

Due to space constraints, we only provide proof sketches.

2 Preliminaries

We review notions from propositional logic and knowledge compilation. For $n \in \mathbb{N}$, let $[n] := \{1, 2, \ldots, n\}$.

In **propositional logic**, a variable v or its negation $\overline{v}$ is called a *literal* l. A *clause* is a disjunction of literals. A conjunction of clauses is a *conjunctive normal form* (CNF). Using Tseitin transformations (Tseitin 1968), any formula can be efficiently translated into CNF.

The set of all variables in a formula F is denoted as $\mathsf{vars}(F)$. A (partial) *assignment* for a set of variables V is a (partial) function $\alpha : V \to \{0, 1\}$ mapping variables to Boolean values. The set of all $2^{|V|}$ complete assignments for V is denoted as $\langle V \rangle$. Given $V \subseteq \mathsf{vars}(F)$ and $\alpha \in \langle V \rangle$, $F[\alpha]$ represents the formula where all variables $x \in V$ are replaced by $\alpha(x)$. An assignment α *satisfies* F if $F[\alpha] = 1$, denoted $\alpha \models F$, and α *falsifies* F if $F[\alpha] = 0$. A formula F is *satisfiable* if there is some assignment $\alpha \in \langle \mathsf{vars}(F) \rangle$ with $\alpha \models F$. Otherwise, the formula is *unsatisfiable*.

Next, we introduce the *model counting problem* #SAT. Throughout the paper, we denote the CNF we want to count on as φ. Given φ, #SAT asks to compute $\#\varphi := |\mathsf{Mod}(\varphi)|$ with $\mathsf{Mod}(\varphi) := \{\alpha \in \langle \mathsf{vars}(\varphi) \rangle \mid \alpha \models \varphi\}$.

For a CNF F and set V of variables, we denote $cc(F, V)$ as the smallest *connected component*. Formally, it is the smallest set of clauses such that for every $C \in F$:

- if $\mathsf{vars}(C) \cap V \neq \emptyset$, then $C \in cc(F, V)$, and
- if there is some clause $D \in cc(F, V)$ with $\mathsf{vars}(C) \cap \mathsf{vars}(D) \neq \emptyset$, then $C \in cc(F, V)$.

We define *semantic consequence*. If for every assignment $\alpha \in \langle \mathsf{vars}(F) \cup \mathsf{vars}(G) \rangle$ holds that $\alpha \models F$ implies $\alpha \models G$, we write $F \models G$. If $F \models G$ and $G \models F$, we write $F \equiv G$.

Any assignment can be seen as a CNF only consisting of unit clauses enforcing the assignment. Since a CNF is a set of clauses, set operations are well defined on CNFs and assignments. Two partial assignments are called *consistent* if they agree on their intersection.

A **proof system** for a language L is a polynomial time computable function f with range $\mathsf{rng}(f) = L$ (Cook and Reckhow 1979). In this paper L is always UNSAT or #SAT. We call π an f-proof for $x \in L$ if $f(\pi) = x$.

We use *simulations* to compare two proof systems P and Q for the same language. We say that P *p-simulates* Q if every Q-proof can be transformed in polynomial time into a P-proof of the same formula. If P and Q p-simulate each other, they are called *p-equivalent*, denoted $P \equiv Q$.

The best-studied proof system for UNSAT is *resolution*. It is a *line-based* proof system with clauses as proof lines. With the *resolution rule* we derive the clause $C \cup D$ from previous clauses $C \cup \{x\}$ and $D \cup \{\overline{x}\}$. A *resolution refutation* of a CNF is a derivation of the empty clause $\square$.

Knowledge compilation has been extensively studied (Darwiche and Marquis 2002; Jukna 2012) and has a tight connections to model counting. Here we only need the notion of a Decision-DNNF.

A *circuit* C is a rooted directed acyclic graph with labelled nodes that we call *gates*. The set of all variables appearing in C is denoted as $\mathsf{vars}(C)$. For any gate $v \in C$, we denote the subcircuit with root v as C_v.

A Decision-DNNF, standing for *decision decomposable negation normal form* (Darwiche 1999), is a circuit D that has one unique gate with in-degree 0 which is its root and represents the output of D. A simple Decision-DNNF is illustrated in Figure 1. Any leaf of D, i.e. any gate with out-degree 0, is a 0-gate labelled with 0 or a 1-gate labelled with 1. Every gate in D that is not a leaf is either an AND-gate or a DECISION-gate.

An AND-gate v is labelled with an $\wedge$ and is *decomposable* which means that for any two child gates v_i and v_j of v, we have $\mathsf{vars}(D_{v_i}) \cap \mathsf{vars}(D_{v_j}) = \emptyset$. For simplicity, we assume

w.l.o.g. that AND-gates have exactly two child nodes. A DECISION-gate is labelled with some variable x that is decided in this gate, i.e. there are two outgoing edges corresponding to the assignments $x = 0$ and $x = 1$. Further, there must not be a path in D from the root to a leaf that decides the same variable more than once.

The main motivation for compiling CNFs to Decision-DNNFs is that on Decision-DNNFs various queries can be handled efficiently, in particular model counting and clause entailment:

Theorem 2.1. *(Darwiche and Marquis 2002) Let D be a Decision-DNNF and φ a CNF, then:*

- *we can compute the model count of D in time $|D|^{O(1)}$,*
- *we can check $D \models \varphi$ in time $(|D| \cdot |\varphi|)^{O(1)}$.*

3 KC^{PS}_{Syn} Proofs Characterise #SAT Solvers

3.1 Model counting algorithms

Many of the state-of-the-art knowledge compilers and model counters operate in a top-down fashion and can be seen as natural generalisations of SAT solvers based on DPLL. Following Capelli, Lagniez, and Marquis (2021), we provide a high-level sketch of this solving approach which we call KC^{Alg}_{Syn}. The pseudocode in Algorithm 1 gives details. Note that actual solver implementations use further heuristics or pre-, in- and postprocessing techniques.

The main idea of the solver is to build a decision tree that branches on variables until the remaining formula is satisfied or unsatisfiable. Soundness of a decision on x holds due to $F \equiv (x \wedge F[x]) \vee (\overline{x} \wedge F[\overline{x}])$ and we observe that $\#F = \#F[x] + \#F[\overline{x}]$. As the algorithm might visit the exact same formula multiple times we use a cache to keep track of formulas already computed. A further improvement can be obtained by checking if the formula consists of independent subformulas, i.e. if $F = F_1 \wedge F_2$ with $\mathsf{vars}(F_1) \cap \mathsf{vars}(F_2) = \emptyset$. In this case, we deal with the two formulas independently and have $\#F = \#F_1 \cdot \#F_2$. Finally, in order to check if the remaining formula is unsatisfiable, we use a SAT solver, typically based on CDCL. Clauses that are learned during a SAT call are saved as they might be helpful for subsequent SAT calls.

3.2 Certifiable Decision-DNNF circuits

The proof system *certifiable Decision-DNNF circuits* was proposed to certify KC^{Alg}_{Syn} (Capelli, Lagniez, and Marquis 2021). For brevity, we refer to it as C-dec-DNNF.

A C-dec-DNNF proof for a CNF φ is a restricted and annotated Decision-DNNF D such that we can verify $D \equiv \varphi$ efficiently. Conceptually, it is very similar to the proof system KC^{PS}_{Syn} which we define later in this section. We omit the formal definition of C-dec-DNNF as the precise details do not matter for the remainder of this exposition. However, the main motivation for the introduction of C-dec-DNNF – its tight connection to KC^{Alg}_{Syn}– is of importance to us and is captured in Theorem 3.1:

Algorithm 1: Pseudocode for KC^{Alg}_{Syn} (Capelli, Lagniez, and Marquis 2021)

procedure COMPILE(F, $L = \emptyset$, $R = \emptyset$)
  Data: • **Input:** CNF F
    • **Input:** set L of literals
    • **Input:** set R of learnt clauses; $F \models R$
  Result: Decision-DNNF $D \equiv F[L]$
  if $F[L]$ has no clause **then return** new 1-gate
  if $F[L]$ has an empty clause **then return** new 0-gate
  if $R[L]$ has an empty clause **then return** new 0-gate
  Call a SAT solver on F with literals in L blocked
  Let R' be the clauses learnt by this call
  $R \leftarrow R \cup R'$
  if UNSAT($F[L]$) **then return** new 0-gate
  if cache($F[L]$) $\neq$ nil **then return** cache($F[L]$)
  $F' \leftarrow \{ C \in F \mid C[L] \text{ is not satisfied} \}$
  if $F' = F_1 \wedge \cdots \wedge F_k$ with $k \geq 2$ and
    $\mathsf{vars}(F_i[L]) \cap \mathsf{vars}(F_j[L]) = \emptyset$ for $i \neq j$ **then**
    $v \leftarrow$ new AND-gate
    for $i \leftarrow 1$ **to** k **do**
      $w_i \leftarrow$ COMPILE(F_i, L, R)
      connect v to w_i
  else
    Choose $x \in \mathsf{vars}(F[L])$
    $v \leftarrow$ new DECISION-gate deciding x
    for $\ell \in \{x, \neg x\}$ **do**
      $w \leftarrow$ COMPILE($F, L \cup \{\ell\}, R$)
      connect v to w with label ℓ
  cache($F[L]$) $\leftarrow v$
  return v

Theorem 3.1. *(Capelli, Lagniez, and Marquis 2021) Let R be a run of KC^{Alg}_{Syn} on a CNF φ. Then we can extract from R a C-dec-DNNF proof of φ in time $O(|R|)$.*

This system was designed for certifying outputs of knowledge compilers. It is not obvious whether it characterises the runtime of KC^{Alg}_{Syn}, i.e. whether the converse of Theorem 3.1 holds. The question is: can KC^{Alg}_{Syn} compute a Decision-DNNF efficiently from a given C-dec-DNNF proof? For the corresponding simulation, we want to illustrate the main problem. Consider the formula $\varphi = a \wedge b \wedge (a \vee c) \wedge (a \vee \overline{c}) \wedge (b \vee c) \wedge (b \vee \overline{c})$. This formula is represented by the Decision-DNNF from Figure 1 and it is possible to construct a C-dec-DNNF proof based on this Decision-DNNF. However, the KC^{Alg}_{Syn} could never construct a circuit like this, as φ cannot be divided into independent subformulas without deciding c first. Thus, the C-dec-DNNF system seems to be slightly stronger than the KC^{Alg}_{Syn} algorithm.

3.3 The new proof system KC^{PS}_{Syn}

Now we introduce our new proof system KC^{PS}_{Syn} which can be seen as a restricted version of C-dec-DNNF such that the problem from Figure 1 disappears. In fact, we show that

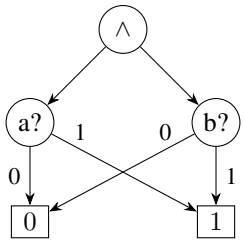

Figure 1: Algorithm KC^{Alg}_{Syn} is not able to find this Decision-DNNF D for $\varphi = a \wedge b \wedge (a \vee c) \wedge (a \vee \overline{c}) \wedge (b \vee c) \wedge (b \vee \overline{c})$ although $D \equiv \varphi$.

KC^{PS}_{Syn} characterises the runtime of KC^{Alg}_{Syn} (Theorem 3.5).

Definition 3.2 (KC^{PS}_{Syn})**.** *Let D be a Decision-DNNF, $\mathcal{F}$ be a labelling that assigns a CNF $\mathcal{F}(v)$ to every gate $v \in D$ and ρ be a set of resolution refutations. A KC^{PS}_{Syn} proof for a CNF φ is a 3-tuple $(D, \mathcal{F}, \rho)$ such that*

(i) $D \models \varphi$,
(ii) for any gate $v \in D$ that is not a 0-gate, $\mathcal{F}(v)$ is satisfiable,
(iii) $\mathcal{F}(r) = \varphi$ for the root $r \in D$,
(iv) for any AND-gate $v \in D$ with children s and t, $\mathcal{F}(s) = cc(\mathcal{F}(v), \mathsf{vars}(D_s))$ and analogously for $\mathcal{F}(t)$. Further, $\mathsf{vars}(\mathcal{F}(s)) \cap \mathsf{vars}(\mathcal{F}(t)) = \emptyset$,
(v) for any DECISION-gate $v \in D$ deciding variable x with children s if $x = 0$ and t if $x = 1$, $\mathcal{F}(s) = \mathcal{F}(v)[\overline{x}]$ and $\mathcal{F}(t) = \mathcal{F}(v)[x]$,
(vi) for any 0-gate $v \in D$ there is a resolution refutation of $\mathcal{F}(v)$ in ρ.

We point out that a KC^{PS}_{Syn} proof does not have to specify $\mathcal{F}$ explicitly as if it exists, it is uniquely determined by D and φ.

Before we further analyse KC^{PS}_{Syn}, we show that is indeed a proof system in the sense of Cook and Reckhow (1979):

Proposition 3.3. *KC^{PS}_{Syn} is a complete and sound proof system for #SAT.*

In fact, the proof of Proposition 3.3 neither uses condition (ii) nor $\mathsf{vars}(\mathcal{F}(s)) \cap \mathsf{vars}(\mathcal{F}(t)) = \emptyset$ in (iv). Thus, KC^{PS}_{Syn} would be a proof system even without these requirements. However, we stick to these additional requirements as the first guarantees the natural invariant $D_v \equiv \mathcal{F}(v)$ for any node $v \in D$ (Proposition 3.4). The second one is needed to show the equivalence to KC^{Alg}_{Syn} (Theorem 3.5).

Proposition 3.4. *Let $(D, \mathcal{F}, \rho)$ be a KC^{PS}_{Syn} proof. Then, for every gate $v \in D$ the circuit D_v is equivalent to $\mathcal{F}(v)$.*

Next, we get to the result that provides the main motivation for the modifications of C-dec-DNNF. That is, we can generate solver runs from KC^{PS}_{Syn} proofs efficiently:

Theorem 3.5. *For each CNF φ, the minimal runtime of KC^{Alg}_{Syn} on φ (with freely choosable heuristics) is polynomially equivalent to the minimal KC^{PS}_{Syn} proof size of φ.*

Proof sketch. We show the two directions separately.

For the *proof extraction*, let R be a run of KC^{Alg}_{Syn} on a CNF φ. By Theorem 3.1, we can extract a C-dec-DNNF proof π based on a Decision-DNNF D efficiently. Next, we can prove that π satisfies the additional property of formula-decomposability, which requires every AND-gate $v \in D$ with children s and t to satisfy $\mathsf{vars}(F_s) \cap \mathsf{vars}(F_t) = \emptyset$. Finally, we can show that KC^{PS}_{Syn} p-simulates formula-decomposable C-dec-DNNF. Thus, we can extract a KC^{PS}_{Syn} proof from π, and therefore also from R, efficiently.

For the other direction, let some KC^{PS}_{Syn} proof $\pi = (D, \mathcal{F}, \rho)$ for a CNF φ be given. We do a depth-first search on D starting at the root and order all gates of D accordingly in a list L. We let algorithm KC^{Alg}_{Syn} run on φ while processing L. Let F be the formula the algorithm considers in the current step and let v be the next gate in L. Then, if F fits to v, i.e. if $F = \mathcal{F}(v)$, then:

- if v is a DECISION-gate, we decide the corresponding variable,
- if v is an AND-gate, we split the current formula accordingly,
- if v is a 0-gate, we let the SAT-solver find the corresponding resolution refutation from ρ.

Otherwise, if $F \neq \mathcal{F}(v)$, then the algorithm has already considered the formula F previously and we can use it from the cache. □

A further insight from the proof is that saving learnt clauses in R can give at most polynomial speed-up. Therefore, from a purely theoretical point of view and ignoring polynomial overhead, clause saving does not help. For practical purposes, of course, it might still help to improve performance.

4 KC^{PS}_{Syn} vs MICE

Next, we compare KC^{PS}_{Syn} to the existing MICE proof system (Fichte, Hecher, and Roland 2022; Beyersdorff, Hoffmann, and Spachmann 2024). We briefly explain MICE and show that it is weaker than KC^{PS}_{Syn}. However, we can strengthen MICE by adding a natural derivation rule. With that rule, the adapted MICE becomes as strong as KC^{PS}_{Syn}.

4.1 The MICE proof system

The proof lines in MICE are called *claims*. A *claim* is a 3-tuple (F, A, c) consisting of a CNF F, a partial assignment A of $\mathsf{vars}(F)$ (called *assumption*) and a count c. Semantically, in our definition a claim (F, A, c) says that $F[A]$ has $c \cdot 2^{|\mathsf{vars}(F[A])|}$ models. This is a slight change compared to the prior definition of MICE, but does not materially impact the proof system. The rules become slightly easier as we get rid of the powers of 2 if the number of variables in the formula changes.

A MICE *proof* of a CNF φ is a sequence of claims $I_1, \ldots, I_k$ that are derived with the inference rules in Figure 2 such that the final claim has the form $(\varphi, \emptyset, c)$ for some

count c. Thus, for the final claim $\#\varphi = c \cdot 2^{|\mathsf{vars}(F)|}$. As the count c in a correct claim (F, A, c) is determined by F and A, we sometimes omit it and write (F, A) instead.

Another small modification in our definition of MICE concerns the rules (0-Ax) and (2-Comp) which are special cases of a more general *composition* derivation rule in Beyersdorff, Hoffmann, and Spachmann (2024). This does not change the strength of the system. However, the relation to Decision-DNNFs becomes clearer.

Axiom. $\dfrac{}{(\emptyset, \emptyset, 1)}$ (Ax)

0-Axiom. $\dfrac{}{(F, A, 0)}$ (0-Ax)

- (A-1) there is a resolution refutation of $F[A]$

2-Composition.

$$\frac{(F, A \wedge \{x = 0\}, c_1) \quad (F, A \wedge \{x = 1\}, c_2)}{(F, A, c_1 + c_2)} \text{ (2-Comp)}$$

- (C-1) $x \notin \mathsf{vars}(A)$

Join.

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

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

Extension.

$$\frac{(F_1, A_1, c_1)}{(F, A, c_1)} \text{ (Ext)}$$

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

Figure 2: Adapted inference rules for MICE, following Beyersdorff, Hoffmann, and Spachmann (2024). Bullet points indicate the conditions under which the rule is applicable.

4.2 A weakness of MICE

It is already known that there exist formulas with polynomial-size Decision-DNNFs while any MICE proof needs exponentially many steps (Beyersdorff et al. 2024). However, it has remained open so far whether MICE allows for efficient proof logging for current solving techniques. It is conceivable that #SAT solvers produce Decision-DNNFs that are restricted enough to extract MICE proofs efficiently. We show that this is not the case, i.e. there exist formulas without short MICE proofs that state-of-the-art solvers can handle efficiently. This follows from our results that KC^{PS}_{Syn} characterises KC^{Alg}_{Syn} solving (Theorem 3.5), while KC^{PS}_{Syn} is stronger than MICE (Theorem 4.1). For the separation we use a variant of the pebbling formulas, well known in proof complexity. More precisely, we use the PEB_n formulas from (Beyersdorff et al. 2024) for which an exponential lower bound for MICE proof size is known (Beyersdorff et al. 2024).

Theorem 4.1. *The PEB_n formulas have KC^{PS}_{Syn} proofs of size $O(n)$ while any MICE proof has size at least $2^{\Omega(n)}$.*

Proof sketch. For the proof it is sufficient to construct KC^{PS}_{Syn} proofs of linear size as the MICE lower bound is already known (Beyersdorff et al. 2024). The key idea for the short KC^{PS}_{Syn} proof is the fact that the restricted formulas are syntactically equivalent under many different partial assignments if we decide the variables in the correct order. □

Therefore, KC^{Alg}_{Syn} with perfectly chosen heuristics is able to handle these separating formulas efficiently while any MICE proof has exponential size.

4.3 Increasing the strength of MICE

Next, we fix this weakness of MICE. As it turns out, this can be done with an additional simple and natural rule. Recall that a MICE claim (F, A, c) states that the formula $F[A]$ is satisfied by exactly the fraction c of all models. Therefore, the following rule seems quite natural: The *Reform* rule allows to derive claim (F_1, A_1) from (F, A) if $F[A]$ and $F_1[A_1]$ are syntactically equivalent formulas. The formal definition of the (Ref) rule is displayed in Figure 3. We refer to MICE with the additional (Ref) rule as MICE_{Ref}. It is easy to verify that MICE_{Ref} is still a valid proof system. Note that for any MICE claim (F, A), we can derive the equivalent claim $(F[A], \emptyset)$ with (Ref) and vice versa.

Reform. $\dfrac{(F_1, A_1, c_1)}{(F, A, c_1)}$ (Ref)

- (R-1) $F_1[A_1]$ and $F[A]$ are syntactically equivalent. That is, they are the exact same set of clauses after removing weakened clauses.

Figure 3: Additional reform rule for MICE.

Next, we show that this augmented MICE system nicely fits into the bigger picture of #SAT proof systems as it is equivalent to some other system we already know, namely KC^{PS}_{Syn}. Therefore, adding the (Ref) rule avoids the shortcoming of MICE illustrated in Theorem 4.1 and closes the gap between the two systems MICE and KC^{PS}_{Syn}.

Theorem 4.2. *MICE_{Ref} is p-equivalent to KC^{PS}_{Syn}.*

Proof sketch. For the first direction, KC^{PS}_{Syn} p-simulates MICE_{Ref}, we use the result that we can extract a Decision-DNNF from a MICE proof efficiently (Beyersdorff, Hoffmann, and Spachmann 2024). This extraction does also work for (Ref) steps. We then show that the resulting Decision-DNNF can be converted into a KC^{PS}_{Syn} proof.

For the other direction, MICE_{Ref} p-simulates KC^{PS}_{Syn}, we consider a KC^{PS}_{Syn} proof $\pi = (D, \mathcal{F}, \rho)$. We associate every gate $v \in D$ with a MICE claim $I_v = (\mathcal{F}(v), \emptyset)$ and show

that we can derive all these claims efficiently in a bottom-up fashion. For that, we distinguish what kind of gate v is:

- If v is a 1-gate, we can derive I_v with (Ax) since $\mathcal{F}(v)$ is a tautological CNF, i.e. $\mathcal{F}(v) = \emptyset$.
- If v is a 0-gate, we can derive I_v with (0-Ax) and use the refutation of $\mathcal{F}(v)$ from ρ as absence of models statement.
- If v is a DECISION-gate deciding variable x, let s be the gate that is reached if $x = 0$ and t if $x = 1$. Per induction hypothesis, we have already derived claims $I_s = (\mathcal{F}(s), \emptyset) = (\mathcal{F}(v)[\overline{x}], \emptyset)$ and $I_t = (\mathcal{F}(t), \emptyset) = (\mathcal{F}(v)[x], \emptyset)$. By applying (Ref) to both claims, we can derive $I'_s = (\mathcal{F}(v), \{\overline{x}\})$ and $I'_t = (\mathcal{F}(v), \{x\})$. Next, we apply (2-Comp) to I'_s and I'_t and obtain $(\mathcal{F}(v), \emptyset) = I_v$.
- If v is an AND-gate with children s and t, we have already derived $I_s = (\mathcal{F}(s), \emptyset)$ and $I_t = (\mathcal{F}(t), \emptyset)$ by the induction hypothesis. Because of condition (iv) of the KC^{PS}_{Syn} system, we have $\mathsf{vars}(\mathcal{F}(s)) \cap \mathsf{vars}(\mathcal{F}(t)) = \emptyset$. Thus, we can apply (Join) to I_s and I_t and obtain claim $(\mathcal{F}(s) \cup \mathcal{F}(t), \emptyset) = (\mathcal{F}(v), \emptyset) = I_v$.

As $\mathcal{F}(r) = \varphi$ for the root r of D by condition (iii), the resulting sequence of MICE claims derives the final claim $(\varphi, \emptyset)$ and is thus indeed a valid MICE proof. □

5 Semantic Caching

With a closer look at algorithm KC^{Alg}_{Syn} a natural optimisation springs to mind: improve caching from a purely syntactic approach to a more semantic one. Although it is not obvious how to do that efficiently in practice, we will investigate this question from a theoretical point of view. That is, we can increase the strength of both KC^{PS}_{Syn} and MICE_{Ref} very naturally if we allow semantic caching and reforming instead of requiring syntactic equivalence of the formulas involved. However, we have to provide some kind of proof for this equivalence.

We start with the formal definitions of the systems.

Definition 5.1 (KC^{PS}_{Sem})**.** *KC^{PS}_{Sem} is a variant of KC^{PS}_{Syn} where conditions (iii), (iv) and (v) are relaxed: syntactic equality of formulas is replaced by semantic equivalence. For each case where formulas F and G are equivalent, resolution refutations of $F \wedge \overline{G}$ as well as $G \wedge \overline{F}$ are additionally required.*

Semantic Reform. $\dfrac{(F_1, A_1, c_1)}{(F, A, c_1)}$ (Ref-Sem)

- (R-1) The MICE proof contains a proof that $F_1[A_1]$ and $F[A]$ are semantically equivalent. That is, there is a resolution refutation of $F_1[A_1 \wedge \overline{C}]$ for every $C \in F[A]$ and of $F[A \wedge \overline{C}_1]$ for every $C_1 \in F_1[A_1]$.

Figure 4: Additional semantic reform rule for MICE.

Algorithm KC^{Alg}_{Sem} is defined similarly to KC^{Alg}_{Syn}. The main difference is that the cache lookup in Line 10 uses a semantic cache that can also return entries that are different from, but semantically equivalent to F. Further, a formula F is considered decomposable if it is *semantically* equivalent to $\bigwedge_{i=1}^{k} F_i$ with $k \geq 2$ and variable-disjoint formulas F_i, which is checked by line 12.

How to implement such a semantic cache efficiently is non-trivial and beyond the scope of this paper. However, we assume that such an implementation would use a heuristic to decide whether to check two formulas for equivalence using a SAT solver.

In the non-deterministic algorithm KC^{Alg}_{Sem} we can therefore assume that the only costs are for successful cache lookups. When looking up F and finding G, this cost is the length of resolution refutations of both $F \wedge \overline{G}$ and $G \wedge \overline{F}$.

Theorem 5.2. *KC^{Alg}_{Sem} is characterized by KC^{PS}_{Sem} (in the same formal sense as in Theorem 3.5).*

Proof sketch. The first direction, extracting a KC^{PS}_{Sem} proof from a KC^{Alg}_{Sem} run, is very similar to the proof extraction in the syntactical setting. Whenever the algorithm uses the cache, it has to provide a resolution proof that shows the semantical equivalence of the current formula and the formula from the cache. We can use the very same resolution proof for the KC^{PS}_{Sem} proof.

The other direction, generating a KC^{Alg}_{Sem} run given a KC^{PS}_{Sem} proof, is also similar to the syntactical case with one small obstacle that arises. That is, in a KC^{PS}_{Sem} proof we can annotate any gate with a semantical equivalent formula while in KC^{Alg}_{Sem} we are not that flexible and can only use these semantically equivalent formulas when we use the cache or decompose the formula. This issue can be resolved by delaying the semantic reformulations. □

We now strengthen MICE and define $\mathsf{MICE}_{Ref\text{-}Sem}$ as MICE augmented by the rule (Ref-Sem) in Figure 4. These semantic variants are natural generalisations of KC^{PS}_{Syn} and MICE_{Ref}. As in Theorem 4.2, the two systems are of equal strength despite their different approaches:

Theorem 5.3. *$\mathsf{MICE}_{Ref\text{-}Sem}$ and KC^{PS}_{Sem} are p-equivalent.*

Proof sketch. The proof is analogous to the syntactical version in Theorem 4.2. □

To put some perspective on the strength of KC^{PS}_{Sem} we compare it to the proof system $\mathsf{CPOG}^{\mathsf{Decision\text{-}DNNF}}$ (Bryant et al. 2023; Beyersdorff et al. 2024) which is special case of CPOG (Bryant et al. 2023):

Definition 5.4 (Bryant et al. (2023), Beyersdorff et al. (2024))**.** *A $\mathsf{CPOG}^{\mathsf{Decision\text{-}DNNF}}$ proof of a CNF φ is a pair $(\mathcal{E}(D), \rho)$ where*

1. *D is a Decision-DNNF with root R and $\mathcal{E}(D)$ a clausal encoding of D such that $D \equiv \varphi$,*
2. *ρ is a resolution refutation of $\varphi \wedge \mathcal{E}(D) \wedge (\overline{\vartheta_R})$.*

Conceptually, a $\mathsf{CPOG}^{\mathsf{Decision\text{-}DNNF}}$ proof for a CNF φ consists of a Decision-DNNF D together with a resolution proof showing that $\varphi \models D$. The other entailment

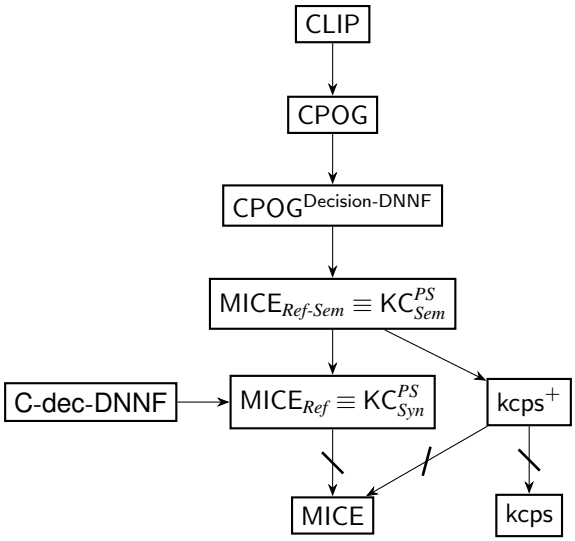

Figure 5: Detailed simulation order of all current #SAT proof systems. A solid edge from A to B indicates that A p-simulates B. If the edge is crossed, A is also exponentially separated from B.

$D \models \varphi$ is easy to check by Theorem 2.1. In the following two propositions we show that KC^{PS}_{Sem} is almost as strong as $\mathsf{CPOG}^{\mathsf{Decision\text{-}DNNF}}$.

Proposition 5.5. $\mathsf{CPOG}^{\mathsf{Decision\text{-}DNNF}}$ *p-simulates* KC^{PS}_{Sem}.

The other direction remains open. The main problem seems to be that $\mathsf{CPOG}^{\mathsf{Decision\text{-}DNNF}}$ can introduce helper variables that could potentially shorten proofs. For instance, it is easy to show that PHP is hard for KC^{PS}_{Sem}, but this appears to be a non-trivial result for $\mathsf{CPOG}^{\mathsf{Decision\text{-}DNNF}}$. However, if we replace the underlying proof system resolution by extended resolution (ER), we can prove the simulation.

Proposition 5.6. KC^{PS}_{Sem} *with ER p-simulates* $\mathsf{CPOG}^{\mathsf{Decision\text{-}DNNF}}$.

Thus, for a CNF φ and any Decision-DNNF $D \equiv \varphi$, we can use D for a KC^{PS}_{Sem} proof, i.e. KC^{PS}_{Sem} is flexible enough to work with arbitrary Decision-DNNFs, a result that does not necessarily hold for the weaker system KC^{PS}_{Syn}.

6 Discussion

In this paper we introduce the new proof systems KC^{PS}_{Syn} together with KC^{PS}_{Sem} and analyse how they relate to known #SAT proof systems. While so far, we only considered MICE and $\mathsf{CPOG}^{\mathsf{Decision\text{-}DNNF}}$, let us conclude by broadening the picture to other existing #SAT proof systems. The first formal proof system for #SAT was kcps (Capelli 2019), later improved to kcps^+ (Capelli 2019; Beyersdorff et al. 2024). As C-dec-DNNF can be seen as an improved and stronger version of these systems, the following result is not surprising:

Proposition 6.1. KC^{PS}_{Sem} *p-simulates* kcps^+.

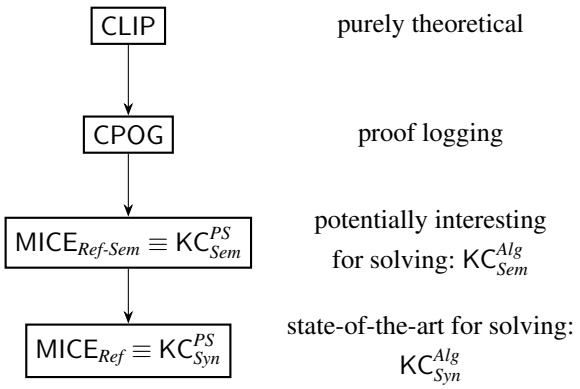

Figure 6: The most relevant proof systems for #SAT (left) and their usage/potential for solving (right).

With that, we obtain the full picture of the current proof systems for #SAT displayed in Figure 5. Let us explain this picture and give some perspectives on what, in our opinion, the most relevant proof systems are.

We advocate the view that the right formulation of MICE is to include the (Ref) rule, which was simply missing in the original definition. This is confirmed by (1) the fact that MICE_{Ref} is equivalent to KC^{PS}_{Syn}, thus leading to a very robust characterisation of the strength of this proof system and (2) its equivalence to state-of-the-art model counting, which was the precise original motivation for the introduction of MICE (Fichte, Hecher, and Roland 2022).

All of kcps, kcps^+ and C-dec-DNNF were suggested as #SAT proof systems close to solvers. In view of Theorem 3.5, we believe that KC^{PS}_{Syn} (which is conceptually close to all the three systems mentioned) is the right answer, further confirmed by the equivalence to MICE_{Ref}. Aesthetically, however, kcps^+ is more pleasing as its definition is simpler, hence kcps^+ could have theoretical interest.

Further, KC^{PS}_{Sem} is *almost* as strong as $\mathsf{CPOG}^{\mathsf{Decision\text{-}DNNF}}$. For both systems, we can use arbitrary Decision-DNNFs to represent the formula on which we count. Should $\mathsf{CPOG}^{\mathsf{Decision\text{-}DNNF}}$ be stronger, then only for propositional reasons, because when changing the underlying proof system from resolution to extended resolution, they become equivalent. Thus, the two systems appear to be quite close to each other (with KC^{PS}_{Sem} potentially being the better choice).

Finally, CLIP (Chede, Chew, and Shukla 2024) is a theoretically elegant, but very powerful proof system, far beyond current solving techniques or proof logging needs.

Thus, in our view, the emerging picture of the most relevant #SAT proof systems together with their potential for solving can be displayed as in Figure 6.

All simulations from Figure 6 are proven, and we also conjecture that all systems can in fact be separated. The separation of KC^{PS}_{Syn} and KC^{PS}_{Sem} appears to be of particular interest, as this would provide insights on whether semantic caching could potentially improve practical model counting.

Acknowledgements

The authors are supported by the Carl-Zeiss Foundation in the project Interactive Inference.

References

Bacchus, F.; Dalmao, S.; and Pitassi, T. 2003. Algorithms and Complexity Results for #SAT and Bayesian Inference. In *FOCS*, 340–351. IEEE Computer Society.

Baluta, T.; Chua, Z. L.; Meel, K. S.; and Saxena, P. 2021. Scalable Quantitative Verification For Deep Neural Networks. In *ICSE*, 312–323. IEEE.

Beyersdorff, O.; Fichte, J. K.; Hecher, M.; Hoffmann, T.; and Kasche, K. 2024. The Relative Strength of #SAT Proof Systems. In *SAT*, volume 305 of *LIPIcs*, 5:1–5:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Beyersdorff, O.; Hoffmann, T.; and Spachmann, L. N. 2024. Proof Complexity of Propositional Model Counting. *J. Satisf. Boolean Model. Comput.*, 15(1): 27–59.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2021. *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications. IOS Press.

Bryant, R. E.; Nawrocki, W.; Avigad, J.; and Heule, M. J. H. 2023. Certified Knowledge Compilation with Application to Verified Model Counting. In *SAT*, volume 271, 6:1–6:20.

Capelli, F. 2017. Understanding the complexity of #SAT using knowledge compilation. In *LICS*, 1–10.

Capelli, F. 2019. Knowledge Compilation Languages as Proof Systems. In *SAT*, volume 11628, 90–99.

Capelli, F.; Lagniez, J.; and Marquis, P. 2021. Certifying Top-Down Decision-DNNF Compilers. In *AAAI*, 6244–6253.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2021. Approximate Model Counting. In *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 1015–1045. IOS Press.

Chede, S.; Chew, L.; and Shukla, A. 2024. Circuits, Proofs and Propositional Model Counting. In *FSTTCS*, volume 323 of *LIPIcs*, 18:1–18:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Cook, S. A. 1971. The complexity of theorem proving procedures. In *Proc. 3rd Annual ACM Symposium on Theory of Computing*, 151–158.

Cook, S. A.; and Reckhow, R. A. 1979. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1): 36–50.

Darwiche, A. 1999. Compiling Knowledge into Decomposable Negation Normal Form. In *IJCAI*, 284–289.

Darwiche, A.; and Marquis, P. 2002. A knowledge compilation map. *JAIR*, 17: 229–264.

Davis, M.; Logemann, G.; and Loveland, D. W. 1962. A machine program for theorem-proving. *Commun. ACM*, 5(7): 394–397.

Davis, M.; and Putnam, H. 1960. A Computing Procedure for Quantification Theory. *J. ACM*, 7(3): 201–215.

Dueñas-Osorio, L.; Meel, K. S.; Paredes, R.; and Vardi, M. Y. 2017. Counting-Based Reliability Estimation for Power-Transmission Grids. In *AAAI*, 4488–4494. AAAI Press.

Fichte, J. K.; and Hecher, M. 2025. The Model Counting Competitions 2021-2023. *CoRR*, abs/2504.13842.

Fichte, J. K.; Hecher, M.; and Hamiti, F. 2021. The Model Counting Competition 2020. *JEA*, 26(1): 1–26.

Fichte, J. K.; Hecher, M.; and Roland, V. 2022. Proofs for Propositional Model Counting. In *SAT*, volume 236, 30:1–30:24.

Huang, J.; and Darwiche, A. 2007. The Language of Search. *J. Artif. Intell. Res.*, 29: 191–219.

Jukna, S. 2012. *Boolean Function Complexity - Advances and Frontiers*, volume 27. ISBN 978-3-642-24507-7.

Lagniez, J.; and Marquis, P. 2017. An Improved Decision-DNNF Compiler. In *IJCAI*, 667–673. ijcai.org.

Latour, A. L. D.; Babaki, B.; Dries, A.; Kimmig, A.; Van den Broeck, G.; and Nijssen, S. 2017. Combining Stochastic Constraint Optimization and Probabilistic Programming - From Knowledge Compilation to Constraint Solving. In *CP*, volume 10416, 495–511. Springer.

Marques Silva, J. P.; Lynce, I.; and Malik, S. 2021. Conflict-Driven Clause Learning SAT Solvers. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications. IOS Press.

Muise, C. J.; McIlraith, S. A.; Beck, J. C.; and Hsu, E. I. 2012. Dsharp: Fast d-DNNF Compilation with sharpSAT. In *AI*, volume 7310 of *Lecture Notes in Computer Science*, 356–361. Springer.

Pipatsrisawat, K.; and Darwiche, A. 2011. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2): 512–525.

Shi, W.; Shih, A.; Darwiche, A.; and Choi, A. 2020. On Tractable Representations of Binary Neural Networks. In *KR*, 882–892.

Thurley, M. 2006. sharpSAT - Counting Models with Advanced Component Caching and Implicit BCP. In *SAT*, volume 4121 of *Lecture Notes in Computer Science*, 424–429. Springer.

Toda, S. 1991. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20: 865–877.

Tseitin, G. C. 1968. On the complexity of derivations in propositional calculus. In Slisenko, A. O., ed., *Studies in Mathematics and Mathematical Logic, Part II*, 115–125.

Wetzler, N.; Heule, M.; and Hunt Jr., W. A. 2014. DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs. In Sinz, C.; and Egly, U., eds., *Theory and Applications of Satisfiability Testing (SAT*, volume 8561 of *Lecture Notes in Computer Science*, 422–429. Springer.

Zhai, E.; Chen, A.; Piskac, R.; Balakrishnan, M.; Tian, B.; Song, B.; and Zhang, H. 2020. Check before You Change: Preventing Correlated Failures in Service Updates. In *NSDI*, 575–589. USENIX Association.