Model Counting for Dependency Quantified Boolean Formulas

Long-Hin Fung[1], Che Cheng[1], Jie-Hong Roland Jiang[1], Friedrich Slivovsky[2], Tony Tan[2]

[1]National Taiwan University [2]University of Liverpool

r12922017@csie.ntu.edu.tw, {f11943097,jhjiang}@ntu.edu.tw, {F.Slivovsky,tonytan}@liverpool.ac.uk

Abstract

Dependency Quantified Boolean Formulas (DQBF) generalize QBF by explicitly specifying which universal variables each existential variable depends on, instead of relying on a linear quantifier order. The satisfiability problem of DQBF is NEXP-complete, and many hard problems can be succinctly encoded as DQBF. Recent work has revealed a strong analogy between DQBF and SAT: k-DQBF (with k existential variables) is a succinct form of k-SAT, and satisfiability is NEXP-complete for 3-DQBF but PSPACE-complete for 2-DQBF, mirroring the complexity gap between 3-SAT (NP-complete) and 2-SAT (NL-complete).

Motivated by this analogy, we study the model counting problem for DQBF, denoted #DQBF. Our main theoretical result is that #2-DQBF is #EXP-complete, where #EXP is the exponential-time analogue of #P. This parallels Valiant's classical theorem stating that #2-SAT is #P-complete. As a direct application, we show that first-order model counting (FOMC) remains #EXP-complete even when restricted to a PSPACE-decidable fragment of first-order logic and domain size two.

Building on recent successes in reducing 2-DQBF satisfiability to symbolic model checking, we develop a dedicated 2-DQBF model counter. Using a diverse set of crafted instances, we experimentally evaluated it against a baseline that expands 2-DQBF formulas into propositional formulas and applies propositional model counting. While the baseline worked well when each existential variable depends on few variables, our implementation scaled significantly better to larger dependency sets.

Code and benchmarks —
https://github.com/Sat-DQBF/sharp2DQR

1 Introduction

There has been tremendous progress in SAT solving over the past few decades, enabling widespread applications across many areas of computing, including reasoning tasks in AI (Biere et al. 2009, 2023; Fichte et al. 2023). However, certain problems in hardware verification and synthesis are unlikely to admit succinct encodings in propositional logic, prompting research into automated reasoning in more expressive logics (Jiang 2009; Balabanov and Jiang 2015; Scholl and Becker 2001; Gitina et al. 2013a; Bloem, Könighofer, and Seidl 2014; Chatterjee et al. 2013; Kuehlmann et al. 2002; Ge-Ernst et al. 2022).

A natural candidate for such applications is the logic of *Dependency Quantified Boolean Formulas* (DQBF), an extension of Quantified Boolean Formulas (QBF) with Henkin quantifiers that annotate each existential variable with a set of universal variables it depends on (Balabanov, Chiang, and Jiang 2014). A model of a DQBF consists of *Skolem functions* that map each existential variable to a truth value based on an assignment to its universal dependencies. The fine-grained control over variable dependencies allows DQBF to naturally express problems such as constrained program synthesis (Golia, Roy, and Meel 2021) and equivalence checking of partially specified circuits (Gitina et al. 2013b). This has led to active research over the past decade and the development of several solvers (Fröhlich et al. 2014; Tentrup and Rabe 2019; Gitina et al. 2015; Wimmer et al. 2017; Síc and Strejcek 2021; Reichl, Slivovsky, and Szeider 2021; Reichl and Slivovsky 2022; Golia, Roy, and Meel 2023), as well as the inclusion of a dedicated DQBF track in recent QBF evaluations (Pulina and Seidl 2019).

While satisfiability is a central question in DQBF, many synthesis and verification tasks benefit from knowing *how many* solutions exist. Counting models can help debug and refine specifications: for instance, an unexpectedly large number of Skolem functions may suggest that the specification admits unintended behaviour. Model counters have been developed for QBF with one quantifier alternation (Plank, Möhle, and Seidl 2024) as well as Boolean synthesis (Shaw, Juba, and Meel 2024), and more recently, for general QBF (Capelli et al. 2024).

In this paper, we consider the model counting problem for DQBF, denoted #DQBF. This is a formidable problem, since even deciding whether a DQBF has a model is NEXP-complete (Peterson and Reif 1979; Chen et al. 2022; Cheng et al. 2025). Moreover, because DQBF allows arbitrary and potentially incomparable dependency sets, existing techniques for #QBF that rely on a linear order of quantifiers cannot be applied.

To support the intuition that #DQBF is a particularly difficult problem, we first prove that even the model counting problem for DQBF with just two existential variables, denoted #2-DQBF, is #EXP-complete. This is despite the fact that satisfiability of 2-DQBF is "only" PSPACE-complete (Fung and Tan 2023). Our proof builds on a recent result on 2-DNF model counting (Bannach et al. 2025) and uses the close correspondence between k-DQBF and k-SAT (Fung and Tan 2023). This hardness result is analogous to the well-known hardness of #2-SAT: while 2-SAT is solvable in polynomial time, counting its models is #P-complete (Valiant 1979a,b).

Note that functions in #EXP may output doubly exponential numbers, which require exponentially many bits. Thus, the standard polynomial time Turing reductions for establishing #P-hardness, as in the case of #2-SAT in (Valiant 1979a,b), is not appropriate for the class #EXP. To circumvent this issue, we introduce a new kind of polynomial-time reduction, called a *poly-monious reduction* (see Section 2 for the definition), which lies between classical parsimonious reductions and polynomial-time Turing reductions. Under poly-monious reductions, #2-SAT is still #P-complete (Bannach et al. 2025).

Another notion of hardness for counting problem requires the reduction to be parsimonious (Ladner 1989). However, since 2-SAT is NL-complete, #2-SAT is not #P-hard under parsimonious reduction, unless NL $=$ NP. We believe that the notion of poly-monious reduction is well-suited for establishing #EXP-hardness, as it strikes a balance between parsimonious reductions and polynomial time Turing reductions in terms of strength.

As an application of the hardness of #2-DQBF, we show that the combined complexity of first-order model counting (FOMC)—a central problem in statistical relational AI—is #EXP-complete (over varying vocabulary). FOMC is defined as given FO sentence Ψ and a number N in unary, compute the number of models of Ψ with domain $\{1, \dots, N\}$. The combined complexity of FOMC is the complexity measured in terms of both the sentence Ψ and the number N. While the #EXP-hardness of FOMC can already be inferred from classical results in logic (Lewis 1980),[1] we obtain a stronger result: FOMC is #EXP-hard even when the domain size is restricted to 2 and the base logic is a PSPACE-decidable fragment of FO.[2] This may help explain why scalable FO model counters have remained elusive despite intensive research efforts for over a decade.

Motivated by our result that #DQBF remains hard even for just two existential variables, we explore the viability of solving #2-DQBF in practice. Due to the double-exponential number of possible Skolem functions, direct enumeration is infeasible. Similarly, expanding a DQBF into a propositional formula leads to an exponential blow-up, rendering state-of-the-art #SAT solvers impractical.

Instead, we build on a recent success in reducing 2-DQBF satisfiability to model checking (Fung et al. 2024) and interpret 2-DQBF instances as succinctly represented implication graphs. Based on this idea, we propose a model counting algorithm that proceeds in two main phases. In the first phase, it constructs a Binary Decision Diagram (BDD) representing reachability in the implication graph. This phase benefits from mature tools developed by the formal methods community, including the IC3 algorithm, the CUDD package for BDD manipulation, and ABC's implementation of exact reachability (Bradley and Manna 2007; Bradley 2011; Eén, Mishchenko, and Brayton 2011; Somenzi 2009; Brayton and Mishchenko 2010). In the second phase, our algorithm counts Skolem functions by analysing each weakly connected component separately—similar in spirit to component-based decomposition in propositional model counters (Gomes, Sabharwal, and Selman 2021). Within each component, it suffices to enumerate Skolem functions for just one existential variable. We further restrict attention to *partial* Skolem functions defined only on the variables local to each component. This avoids explicit enumeration and enables us to handle instances with up to $2^{2^{64}}$ Skolem functions. The techniques used in this phase combine new ideas with existing methods (Reichl, Slivovsky, and Szeider 2021; Fung et al. 2024).

We evaluate our implementation on a diverse set of crafted benchmarks. As a baseline, we use a pipeline that expands a DQBF to a propositional formula and applies the #SAT solver Ganak (Sharma et al. 2019). While Ganak performs well on some smaller instances, its reliance on explicit expansion becomes a bottleneck as dependency sets grow. In contrast, our solver scales gracefully and consistently outperforms the baseline on DQBF with larger dependency sets.

We also performed experiments with state-of-the-art FO model counters. While our approach can only be applied to FOMC with binary relations, this is enough to encode problems such as counting the number of independent sets in highly symmetric graphs. In some cases, our implementation was able to handle instances with more than 2^{127} solutions, far beyond the practical reach of current FO model counters. This indicates that an analogue of our component decomposition technique for #2-DQBF may improve FO model counters in restricted, highly symmetric settings.

Related work. FOMC is often studied in data complexity setting, i.e., the FO sentence is fixed and the complexity is measured only in terms of the domain size. It is shown in (Beame et al. 2015) that there is a three-variable sentence such that the data complexity of its FOMC is #P_1-complete. For two-variable fragment, the data complexity drops to PTIME (Tóth and Kuzelka 2024; van Bremen and Kuzelka 2023; Beame et al. 2015). The combined complexity is #P-complete, but assuming that the vocabulary is fixed (Beame et al. 2015). A tightly related problem to FOMC is query evaluation on probabilistic database, whose combined complexity is #P-complete (Dalvi and Suciu 2004), but again, under the assumption of fixed vocabulary.

The notion of combined and data complexity was introduced in (Vardi 1982) in the context of database query evaluation, to better understand which component (the query/the data/both) contributes more to the complexity of query evaluation. Since then, as hinted in the previous paragraph, it

[1] A close inspection of the proof in (Lewis 1980) shows poly-monious reductions from languages in NEXP to the Bernays-Schönfinkel-Ramsey fragment of FO, whose satisfiability problem is known to be NEXP-complete.

[2] In general, the satisfiability problem for FO is undecidable (Trakhtenbrot 1950).

has become the standard notion for establishing fine-grained complexity results for problems involving a few parameters.

2 Preliminaries

Notation. Let $\mathbb{B} = \{\bot, \top\}$, where $\bot$ and $\top$ denote the Boolean false and true values. A literal is either a Boolean variable or its negation. We write $x^\top$ to denote the literal x and $x^\bot$ to denote $\neg x$. The sign of the literal x^b is the bit b.

We use the symbols a, b, c to denote elements in $\mathbb{B}$, and the bar version $\bar{a}, \bar{b}, \bar{c}$ to denote strings in $\mathbb{B}^*$ with $|\bar{a}|$ denoting the length of $\bar{a}$. Boolean variables are denoted by x, y, z, u, v and the bar version $\bar{x}, \bar{y}, \bar{z}, \bar{u}, \bar{v}$ denote vectors of Boolean variables with $|\bar{x}|$ denoting the length of $\bar{x}$. We insist that in a vector $\bar{x}$ there is no variable occurring more than once. Abusing the notation, we write $\bar{z} \subseteq \bar{x}$ to denote that every variable in $\bar{z}$ also occurs in $\bar{x}$.

As usual, $\varphi(\bar{x})$ denotes a Boolean formula with variables $\bar{x}$. When it is clear from the context, we simply write φ. For $\bar{z} \subseteq \bar{x}$ and $\bar{a} \in \mathbb{B}^*$ where $|\bar{a}| = |\bar{z}|$, $\varphi[\bar{z}/\bar{a}]$ denotes the formula obtained from φ by assigning the values in $\bar{a}$ to $\bar{z}$. Obviously, if $\bar{z} = \bar{x}$, then $\varphi[\bar{z}/\bar{a}]$ is either $\bot$ or $\top$.

Poly-monious reductions. Let Σ be a finite alphabet. A *poly-monious reduction* from a function $F : \Sigma^* \to \mathbb{N}$ to another function $G : \Sigma^* \to \mathbb{N}$ is a polynomial time deterministic Turing machine M together with a polynomial $p(s_1, \ldots, s_t)$ such that on input word w, M outputs t strings $v_1, \ldots, v_t$ where $F(w) = p(G(v_1), \ldots, G(v_t))$.

Note that poly-monious reductions are a slight generalization of the classical parsimonious and c-monious reductions, but weaker than polynomial time Turing reductions. Parsimonious reduction is a poly-monious reduction with the identity polynomial $p(s) = s$. The c-monious reduction (Bannach et al. 2025) is a poly-monious reduction with the polynomial $p(s) = cs$. When restricted to functions in #P, a poly-monious reduction with polynomial $p(s_1, \ldots, s_t)$ is a special case of polynomial time Turing reduction in the sense that the number of calls to the oracle is fixed to t, which does not depend on the input word.

#EXP-complete functions. A function $F : \Sigma^* \to \mathbb{N}$ is in #EXP, if there is a non-deterministic exponential time Turing machine M such that for every word $w \in \Sigma^*$, $F(w)$ is the number of accepting runs of M on w. It is #EXP-*hard*, if for every function $G \in$ #EXP, there is a poly-monious reduction from G to F. Finally, it is #EXP-*complete*, if it is in #EXP and #EXP-hard.

Dependency Quantified Boolean Formulas (DQBF). A *dependency quantified Boolean formula* (DQBF) in prenex normal form is a formula of the form:

$$\Psi \ := \ \forall \bar{x}\, \exists y_1(\bar{z}_1) \cdots \exists y_k(\bar{z}_k) \ \ \psi \tag{1}$$

where $\bar{x} = (x_1, \ldots, x_n)$, each $\bar{z}_i \subseteq \bar{x}$ and ψ, called the *matrix*, is a quantifier-free Boolean formula using variables in $\bar{x} \cup \{y_1, \ldots, y_k\}$. We call $\bar{x}$ the *universal variables*, $y_1, \ldots, y_k$ the *existential variables*, and each $\bar{z}_i$ the *dependency set* of y_i. A k-DQBF is a DQBF with k existential variables. For convenience, we sometimes write $\exists y_i(\bar{z}_i)$ as $\exists y_i(I_i)$ where I_i is the set of indices of the variables in $\bar{z}_i$.

A DQBF Ψ as in (1) is *satisfiable* if there is a tuple $(f_1, \ldots, f_k)$, called *Skolem functions*, such that, for every $1 \le i \le k$, f_i is a formula using only variables in $\bar{z}_i$, and by replacing each y_i with f_i, the matrix ψ becomes a tautology. We call the tuple $(f_1, \ldots, f_k)$ a *solution* or *model* of Ψ and write $(f_1, \ldots, f_k) \models \Psi$. We refer to Ψ as a *uniform DQBF* if for every model $(f_1, \ldots, f_k) \models \Psi$, $f_1, \ldots, f_k$ represent the same Boolean function, i.e., $|\bar{z}_1| = \cdots = |\bar{z}_k| = m$ and for every $\bar{a} \in \mathbb{B}^m$, $f_1(\bar{a}) = \cdots = f_k(\bar{a})$. We write $\#\Psi$ to denote the number of Skolem functions of Ψ.

The *model counting* problem for DQBF, denoted #DQBF, is to compute $\#\Psi$ for a given DQBF Ψ. Its restriction to k-DQBF is denoted by #k-DQBF.

DQBF expansion. We first recall the definition of the expansion of a DQBF from (Fung and Tan 2023), which shows that a DQBF represents an exponentially large CNF formula. We will need an additional notation. For $\bar{z} \subseteq \bar{x}$ and $\bar{a} \in \Sigma^{|\bar{x}|}$, we write $\bar{a}\big|_{\bar{x}\downarrow\bar{z}}$ to denote the projection of $\bar{a}$ to the components in $\bar{z}$ according to the order of the variables in $\bar{x}$. For example, if $\bar{x} = (x_1, \ldots, x_5)$ and $\bar{z} = (x_1, x_2, x_5)$, then $\bot\bot\top\bot\top\big|_{\bar{x}\downarrow\bar{z}}$ is $\bot\bot\top$, i.e., the projection of $\bot\bot\top\bot\top$ to its 1^{st}, 2^{nd} and 5^{th} bits.

Let Ψ be as in Eq. (1). For each $1 \le i \le k$ and for each $\bar{c} \in \mathbb{B}^{|\bar{z}_i|}$, let $X_{i,\bar{c}}$ be a variable. For each $(\bar{a}, \bar{b}) \in \mathbb{B}^n \times \mathbb{B}^k$, where $\bar{a} = (a_1, \ldots, a_n)$ and $\bar{b} = (b_1, \ldots, b_k)$, define the clause $C_{\bar{a},\bar{b}} \ := \ X_{1,\bar{c}_1}^{\neg b_1} \vee \cdots \vee X_{k,\bar{c}_k}^{\neg b_k}$, where $\bar{c}_i = \bar{a}\big|_{\bar{x}\downarrow\bar{z}_i}$, for each $1 \le i \le k$. The expansion of Ψ, denoted by $\exp(\Psi)$, is the following k-CNF formula.

$$\exp(\Psi) \ := \bigwedge_{(\bar{a},\bar{b}) \text{ s.t. } \psi[(\bar{x},\bar{y})/(\bar{a},\bar{b})]=\bot} C_{\bar{a},\bar{b}} \tag{2}$$

It is known that Ψ is satisfiable if and only if its expansion $\exp(\Psi)$ is satisfiable (cf. Fung and Tan 2023). More precisely, a solution $(f_1, \ldots, f_k) \models \Phi$ corresponds uniquely to a satisfying assignment of $\exp(\Phi)$, where $X_{i,\bar{c}} = f_i(\bar{c})$ for every $1 \le i \le k$ and $\bar{c} \in \mathbb{B}^{|\bar{z}_i|}$.

3 Complexity of #DQBF

In this section, we will analyse the complexity of #DQBF, starting with #3-DQBF. It is straightforward that #3-DQBF is in #EXP. It is #EXP-hard since every language in NEXP can be reduced parsimoniously in polynomial time to 3-DQBF (Fung and Tan 2023). This gives us the following theorem.

Theorem 1. *#3-DQBF is #EXP-complete.*

Theorem 1 is not surprising, given that the satisfiability problem for 3-DQBF is already NEXP-complete. We will strengthen it by showing that #EXP-hardness already holds for 2-DQBF, whose satisfiability problem is PSPACE-complete.

Before we can prove this, we need to introduce some further terminology. First, we recall the notion of succinct representation of graph introduced in (Galperin and Wigderson 1983). In such a representation, instead of being given the list of edges in a graph, we are given a boolean circuit $C(\bar{x}, \bar{y})$, where $\bar{x}, \bar{y}$ are vectors of Boolean variables of length n. The circuit C represents a graph G_C where the set of vertices is $\mathbb{B}^n$ and there is an edge oriented from $\bar{a}$ to $\bar{b}$, denoted $\bar{a} \to \bar{b}$, iff $C(\bar{a}, \bar{b}) = \top$.

We will interpret a 2-CNF formula F as a directed graph, called the *implication graph of* F, where each clause $(\ell_1 \vee \ell_2)$ represents two edges $(\neg\ell_1 \to \ell_2)$ and $(\neg\ell_2 \to \ell_1)$. If n is the number of variables in F, each literal can be encoded as a binary string $a_0 a_1 \cdots a_{\log n} \in \mathbb{B}^{1+\log n}$, where a_0 is the sign and $a_1 \cdots a_{\log n}$ is the name of the variable.

Finally, we need the notion of *projection* introduced in (Skyum and Valiant 1985). Intuitively, a projection is a special kind of polynomial time reduction where each bit j in the output is determined either by the length of the input or by bit i in the input where the index i can be computed efficiently from index j and the length of the input.

We recall the following lemma from (Fung and Tan 2023), which is inspired by the result in (Papadimitriou and Yannakakis 1986).

Lemma 2. *(Fung and Tan 2023) Suppose there is a projection $\mathcal{A}$ that takes as input a CNF formula and outputs a graph. Then, there is a polynomial time algorithm that transforms a DQBF instance Ψ to a circuit C that succinctly represents the graph $\mathcal{A}(\exp(\Psi))$.*

Using Lemma 2, we can prove the following.

Lemma 3. *Suppose there is a projection $\mathcal{A}$ that takes as input a CNF formula and outputs a 2-CNF formula. Then, there is a polynomial time algorithm $\mathcal{B}$ that transforms a DQBF instance Ψ to a 2-DQBF instance Φ such that $\#\Phi = \#\mathcal{A}(\exp(\Psi))$.*

Proof. Viewing 2-CNF formula as a graph and applying Lemma 2, there is a polynomial time algorithm $\mathcal{A}^*$ that transforms a DQBF Ψ to a circuit C that succinctly represents the implication graph of $\mathcal{A}(\exp(\Psi))$.

The desired algorithm $\mathcal{B}$ works as follows. Let Ψ be the input DQBF. First, run $\mathcal{A}^*$ on Ψ to obtain the circuit $C(u, \bar{x}, u', \bar{x}')$, where $\bar{x}, \bar{x}'$ encode the names of variables and u, u' represent the signs of literals. Then, output the 2-DQBF $\Phi := \forall\bar{x}\forall\bar{x}'\ \exists y_1(\bar{x})\exists y_2(\bar{x}')\ \alpha \wedge \beta$, where

$$\alpha := (\bar{x} = \bar{x}') \to (y_1 = y_2)$$
$$\beta := \bigwedge_{b,b' \in \mathbb{B}} C(b, \bar{x}, b', \bar{x}') \leftrightarrow (y_1^b \to y_2^{b'})$$

Intuitively, α states that Φ is a uniform DQBF and β states that the implication graph of the expansion must have the same edges as G_C.

We claim that $\#\Phi = \#\mathcal{A}(\exp(\Psi))$, i.e., $\#\Phi$ is precisely the number of solutions of the 2-CNF formula represented by the circuit C. By the definition of β, $(b, \bar{a}) \to (b', \bar{a}')$ is an edge in the graph G_C iff a clause $X^b_{1,\bar{a}} \to X^{b'}_{2,\bar{a}'}$ is in $\exp(\Phi)$. Since α states that Skolem functions for y_1, y_2 must be the same, the indices 1 and 2 in the literals $X^b_{1,\bar{a}}$ and $X^{b'}_{2,\bar{a}'}$ can be dropped. It is equivalent to saying that $(b, \bar{a}) \to (b', \bar{a}')$ is an edge in the graph G_C iff a clause $X^b_{1,\bar{a}} \to X^{b'}_{1,\bar{a}'}$ is in $\exp(\Phi)$. Therefore, $\#\Phi = \#\mathcal{A}(\exp(\Psi))$. $\square$

Lemma 4. *There is a polynomial-time reduction that transforms a DQBF Ψ into two 2-DQBFs Φ_1 and Φ_2 such that $\#\Psi = \#\Phi_1 - \#\Phi_2$.*

Proof. It is shown in (Bannach et al. 2025) that there is a polynomial-time reduction that takes as input a CNF formula F and outputs two 2-CNF formulas F_1 and F_2 such that $\#F = \#F_1 - \#F_2$. We observe that their reduction is in fact a projection. Using Lemma 3, we obtain the desired reduction. $\square$

The proof of Lemma 4 is non-constructive. We can strengthen it by giving an explicit reduction that runs in almost linear time, as stated in Lemma 5. The run time is quadratic in the number of existential variables and linear in the length of the matrix.

Lemma 5. *There is a reduction that transforms a DQBF Ψ into two 2-DQBF Φ_1 and Φ_2 such that $\#\Psi = \#\Phi_1 - \#\Phi_2$. The reduction runs in time $O(k^2|\psi|)$, where k is the number of existential variables in Ψ and ψ is the matrix of Ψ.*

Using Lemma 4 or Lemma 5, we obtain the following theorem.

Theorem 6. *#2-DQBF is #*EXP*-complete.*

We can also show that every k-DQBF can be reduced parsimoniously to uniform k-DQBF, which gives us the following corollary.

Corollary 7. *For every $k \geq 2$, #k-DQBF is #*EXP*-complete, even when restricted to uniform k-DQBF.*

4 First-order Model Counting (FOMC)

In this section, we show a tight connection between #DQBF and FOMC. Recall that FOMC is defined as given FO sentence Ψ and a number N in unary, compute the number of models of Ψ with domain $\{1, \ldots, N\}$. We denote by FOMC_{bin} when the number N is given in binary.

It is implicit in (Chen et al. 2022) that FOMC_{bin} can be reduced to #DQBF and that the reduction is parsimonious. We will describe the idea here by an example. Consider the well known smoker-friend example for Markov Logic Networks (Richardson and Domingos 2006):

$$\begin{aligned}\Psi\ :=\ & \forall u\ \texttt{stress}(u) \to \texttt{smoke}(u)\\ & \wedge\ \forall u \forall v\ \texttt{friend}(u,v) \wedge \texttt{smoke}(u) \to \texttt{smoke}(v)\end{aligned}$$

For every n, we will show to construct a DQBF Φ_n such that $\#\Phi_n$ is exactly the number of models of Ψ of size 2^n.

The idea is to represent each of the predicates `stress`, `smoke` and `friend` with a Skolem function. We have $2n$ universal variables $\bar{x}_1, \bar{x}_2$ in Φ_n. The first block of n variables $\bar{x}_1$ corresponds to u and the second block $\bar{x}_2$ corresponds to v. It has 4 existential variables, y_1, y_2, y_3, y_4, corresponding to 4 atoms $\texttt{stress}(u)$, $\texttt{smoke}(u)$, $\texttt{friend}(u,v)$ and $\texttt{smoke}(v)$. The dependency sets are $\bar{x}_1$, $\bar{x}_1$, $\bar{x}_1 \cup \bar{x}_2$ and $\bar{x}_2$, respectively. The matrix of Φ_n is obtained by replacing each atom in Ψ with its corresponding existential variable. Formally,

$$\Phi_n\ := \forall\bar{x}_1\forall\bar{x}_2\ \exists y_1(\bar{x}_1)\exists y_2(\bar{x}_1)\exists y_3(\bar{x}_1, \bar{x}_2)\exists y_4(\bar{x}_2)\ \phi,$$

where

$$\phi := (y_1 \to y_2) \wedge (y_3 \wedge y_2 \to y_4) \wedge (\bar{x}_1 = \bar{x}_2 \to y_2 = y_4)\,.$$

The first two conjuncts correspond to the quantifier-free parts in Ψ. The last conjunct states that y_2 and y_4 must be the same function, since they are intended to represent the same predicate `smoke`. It is not difficult to show that $\#\Phi_n$ is precisely the number of models of Ψ with size 2^n.

Next, we show that FOMC is already hard even when the domain size is fixed to 2.

Theorem 8. *FOMC is $\#$EXP-hard even when the domain size is fixed to 2.*

Proof. The reduction is from uniform 2-DQBF, which is $\#$EXP-hard, by Corollary 7. We fix a uniform 2-DQBF $\Phi := \forall\bar{x}\exists y_1(I)\exists y_2(J)\phi$, where $\bar{x} = (x_1, \dots, x_n)$, $I = \{i_1, \dots, i_m\}$ and $J = \{j_1, \dots, j_m\}$. Let S be a predicate symbol with arity m and U be a unary predicate. Define the FO sentence $\Psi := \exists u_0 \exists u_1 \forall v_1 \cdots \forall v_n\, U(u_1) \wedge \neg U(u_0) \wedge \psi$, where ψ is the formula obtained from ϕ by replacing: (i) each x_i in ϕ with $U(v_i)$ for every $1 \leq i \leq n$; and (ii) y_1 and y_2 with $S(v_{i_1}, \dots, v_{i_m})$ and $S(v_{j_1}, \dots, v_{j_m})$, respectively. The intention is that the Boolean values $\top$ and $\perp$ are represented with membership in the predicate U. A Skolem function $f : \mathbb{B}^m \to \mathbb{B}$ is represented by the relation S. We can show that $\#\Phi$ is half the number of models of Ψ with domain $\{1, 2\}$. $\square$

We can show that the logic required for $\#$EXP-hardness has satisfiability problem decidable in PSPACE, which gives us the following corollary.

Corollary 9. *There is a fragment $\mathcal{L}$ of FO of which the satisfiability problem is in* PSPACE*, but its corresponding FOMC is $\#$EXP-complete even when the domain size is restricted to* 2.

5 Algorithm for $\#$2-DQBF

In this section, we present an algorithm for $\#$2-DQBF that builds on recent advances in 2-DQBF satisfiability checking using symbolic reachability (Fung et al. 2024). The key idea is to interpret the matrix of a 2-DQBF as a succinct encoding of the implication graph induced by its expansion. Our algorithm symbolically decomposes this graph into its weakly connected components and computes the model count by processing each component independently.

We fix the input 2-DQBF $\Phi := \forall\bar{x}\exists y_1(\bar{z}_1)\exists y_2(\bar{z}_2)\ \varphi$. Instead of computing $\#\Phi$ directly, we will compute $\#\exp(\Phi)$. There is a difference because a variable $X_{i,\bar{c}}$ may not even occur in $\exp(\Phi)$, indicating that the Skolem function of y_i is completely unconstrained at assignment $\bar{c}$. We call a variable $X_{i,\bar{c}}$ a *support* variable if it appears in $\exp(\Phi)$; otherwise, it is called *non-support*. Since non-support variables can be assigned arbitrarily, it is sufficient to compute the number of solutions that assign non-support variables to a fixed value, say, $\perp$. We call such solutions *essential solutions*.

Counting non-support variables. $\#\Phi$ can be recovered from the number of essential solutions by multiplying it with 2^m, where m is the number of non-support variables. The set of support/non-support variables can be characterised with Boolean formulas as follows.

Lemma 10. *Let $\mathcal{S}_1 := \{\bar{c} : \neg\varphi[\bar{z}_1/\bar{c}]$ is satisfiable$\}$ and $\mathcal{S}_2 := \{\bar{c} : \neg\varphi[\bar{z}_2/\bar{c}]$ is satisfiable$\}$. The set of support variables in $\exp(\Phi)$ is $\{X_{1,\bar{c}} : \bar{c} \in \mathcal{S}_1\} \cup \{X_{2,\bar{c}} : \bar{c} \in \mathcal{S}_2\}$. Moreover, the number of support and non-support variables is $|\mathcal{S}_1| + |\mathcal{S}_2|$ and $(2^{|\bar{z}_1|} - |\mathcal{S}_1|) + (2^{|\bar{z}_2|} - |\mathcal{S}_2|)$, respectively.*

Given a BDD for the negated matrix $\neg\varphi$, Lemma 10 can be used to efficiently compute the number of support variables $|\mathcal{S}_i|$ by projecting out variables not in $\bar{z}_i$ and counting satisfying assignments.

Overview of the algorithm. In the following, let G_Φ be the implication graph of $\exp(\Phi)$. Let $\widetilde{G}_\Phi$ be the undirected graph obtained from G_Φ by ignoring the edge orientation and adding an edge between a literal and its negation, for every literal in $\exp(\Phi)$. The connected components of $\widetilde{G}_\Phi$ correspond to a partition of the clauses in $\exp(\Phi)$ where no two components share common variables.

High-level pseudocode is shown as Algorithm 1. First, using the reduction in (Fung et al. 2024), we convert Φ to a transition system (I, T), where I is the formula for the initial states and T is the formula for the transition relation, a brief summary of this transformation can be found in the supplementary material. From (I, T), we can deduce whether Φ is satisfiable by constructing a formula φ_{tr} that represents the transitive closure of G_Φ via BDD-based reachability. If it is not satisfiable, the algorithm immediately returns 0.

Now, suppose Φ is satisfiable. From φ_{tr}, we can also construct the formula for $\widetilde{G}_\Phi$. Algorithm 1 iterates through every connected component C in $\widetilde{G}_\Phi$ that contains only the support variables. In each iteration, it computes N_C, the number of assignments on the variables in C that respect the implications in C. For example, if there is an edge $\ell_1 \to \ell_2$ in G_Φ, when ℓ_1 is assigned to $\top$, ℓ_2 must also be assigned to $\top$. If there are k connected components $C_1, C_2, \dots, C_k$ (that contains only support variables), then the number of essential solutions is the product $\prod_{1 \leq i \leq k} N_{C_i}$, since no two components share the same variable.

Counting over a component. This is the most technical part of the algorithm. We start with the following lemma on efficient model counting for 1-DQBF.

Lemma 11. *Let $\Upsilon := \forall\bar{u}\exists y(\bar{v})\ \varphi$ be a satisfiable 1-DQBF.*

- *The number of Skolem functions for Υ is 2^m, where $m = 2^{|\bar{v}|} - |\{\bar{c} : \neg\varphi[\bar{v}/\bar{c}]$ is satisfiable$\}|$.*
- *In particular, for a set $S \subseteq \mathbb{B}^{|\bar{v}|}$, the number of Skolem functions for Υ that differ on S is 2^m, where $m = |S| - |\{\bar{c} : \neg\varphi[\bar{v}/\bar{c}] \wedge (\bar{c} \in S)$ is satisfiable$\}|$.*

Lemma 11 tells us that if we have a candidate Skolem function f for y_1, by substituting y_1 with f, we obtain a 1-DQBF instance Φ' of which the number of Skolem functions restricted to a component can be computed efficiently via Lemma 11. We perform this for every candidate Skolem function for y_1 to compute the number N_C.

Algorithm 1: Count the number of essential solutions for Φ.

1: Transform Φ to a symbolic reachability instance (I, T) using the transformation in (Fung et al. 2024).
2: **if** Φ is unsatisfiable **then**
3:     **return** 0
4: $R \leftarrow$ the set of all support variables.
5: $N \leftarrow 1$.
6: **while** $R \neq \emptyset$ **do**
7:     Pick an arbitrary variable $X_{i,\bar{c}}$ from R.
8:     $C \leftarrow$ the connected component in $\widetilde{G}_\Phi$ that contain $X_{i,\bar{c}}$.
9:     $N_C \leftarrow$ the number of assignments on the variables in C that respect the implications in C.
10:     Remove all the variables in C from R.
11:     $N \leftarrow N \times N_C$.
12: **return** N.

The main challenge is to enumerate all possible Skolem functions for y_1. To do so, we combine the candidate Skolem function enumeration technique in (Reichl, Slivovsky, and Szeider 2021) and the Skolem function extraction in (Fung et al. 2024). Suppose we already have a list of Skolem functions $F = \{f^{(1)}, \ldots, f^{(t)}\}$ for y_1, where each $f^{(i)}$ is given as a Boolean formula. To find a Skolem function different from all functions in this list, it must differ from each $f^{(i)}$ at some $\bar{v}_i$. Let A be a Boolean formula, over variables $\bar{v}_1, \ldots, \bar{v}_t$ where each $|\bar{v}_i| = |\bar{z}_1|$, maintained throughout the enumeration process. We will use A to represent the assignments we can choose to differ from each $f^{(i)}$. The intuition is that if $\mathtt{M}$ is a satisfying assignment for A, then we want to find a function f with $f(\mathtt{M}(\bar{v}_i)) = \neg f^{(i)}(\mathtt{M}(\bar{v}_i))$ for every $1 \leq i \leq t$.

How can we construct the Boolean formula that defines the function f? Here we employ the technique from (Fung et al. 2024). First, we "force" the variable $X_{1,\mathtt{M}(\bar{v}_i)}$ to be assigned with $\neg f^{(i)}(\mathtt{M}(\bar{v}_i))$ by adding the edge

$$X_{1,\mathtt{M}(\bar{v}_i)}^{f^{(i)}(\mathtt{M}(\bar{v}_i))} \to X_{1,\mathtt{M}(\bar{v}_i)}^{\neg f^{(i)}(\mathtt{M}(\bar{v}_i))}$$

into the transition relation T, for every $1 \leq i \leq t$. We then check whether in the transition relation T there is a cycle that contains contradicting literals. If there is such a cycle, we move to the next satisfying assignment of A by "blocking" the assignment $\mathtt{M}$ in A. If there is no such a cycle, we extract the function f for y_1 by employing the technique from (Fung et al. 2024), add f into F, and update A by conjoining it with $C_1[\bar{z}_1/\bar{v}_{t+1}]$, where $\bar{v}_{t+1}$ are fresh variables, and C_1 is a Boolean formula extracted from C that specifies only the literals associated with y_1.

Without additional constraints, we would enumerate many satisfying assignments $\mathtt{M}$ of A which do not lead to a Skolem function. For instance, if $\mathtt{M}(\bar{v}_i) = \mathtt{M}(\bar{v}_j) = \bar{a}$, but $f_i(\bar{a}) \neq f_j(\bar{a})$, there clearly is no function f_{t+1} such that both $f_{t+1}(\bar{a}) \neq f_i(\bar{a})$ and $f_{t+1}(\bar{a}) \neq f_j(\bar{a})$. Such cases, and many more, can be excluded by conjoining A with

$$\bigwedge_{1 \leq i \leq t} \neg\varphi_{tr}\left(X_{1,\bar{v}_{t+1}}^{\neg f^{(t+1)}(\bar{v}_{t+1})}, X_{1,\bar{v}_j}^{f^{(j)}(\bar{v}_j)}\right). \tag{3}$$

Recall that φ_{tr} is a formula that represents the edges of the transitive closure of the implication graph. The formula $\varphi_{tr}\left(X_{1,\bar{z}^{(i)}}^{\neg f^{(i)}(\bar{z}^{(i)})}, X_{1,\bar{z}^{(j)}}^{f^{(j)}(\bar{z}^{(j)})}\right)$ represents the formula obtained by substituting the variable representing the two literals in φ_{tr} with $X_{1,\bar{z}^{(i)}}^{\neg f^{(i)}(\bar{z}^{(i)})}, X_{1,\bar{z}^{(j)}}^{f^{(j)}(\bar{z}^{(j)})}$.

Algorithm 2: Counting N_C

1: Let C_1, C_2 be the literals in C associated with y_1, y_2, resp.
2: $F, A, n \leftarrow \emptyset, \top, 0$;
3: $T' \leftarrow T$ ▷ T is the transition relation constructed from Φ.
4: **while** A is satisfiable **do**
5:     Let $\mathtt{M}$ be a satisfying assignment of A
    ▷ Force the candidate to be different from the previous ones
6:     $T' \leftarrow T' \wedge \textsc{ForceAssignment}(\mathtt{M}, F)$
    ▷ Check if such assignments leads to no Skolem functions
7:     $E' \leftarrow \textsc{ComputeReachable}(E, T')$
8:     **if** $\textsc{CheckBadCycle}(E')$ **then**
9:         $A \leftarrow A \wedge \textsc{BlockAssignment}(\mathtt{M})$
10:         **continue**
    ▷ Count the number of Skolem functions and update A
11:     $f \leftarrow \textsc{ComputeValidCandidate}(T')$
12:     $n \leftarrow n + \textsc{Count1DQBFOnComponent}(f)$
13:     $F \leftarrow F \cup \{f\}$
14:     $\textsc{Update}(A)$
15: **return** n ▷ n is the number N_C

The intended meaning of Eq. (3) is as follows. The conjunct $C_1[\bar{z}_1/\bar{v}_{t+1}]$ states that the place $\bar{v}$ where the next Skolem function differs from f must be in component C_1. Each conjunct $\neg\varphi_{tr}\left(X_{1,\bar{z}^{(i)}}^{\neg f^{(i)}(\bar{z}^{(i)})}, X_{1,\bar{z}^{(j)}}^{f^{(j)}(\bar{z}^{(j)})}\right)$ ensures that the edge $X_{1,\bar{z}^{(i)}}^{\neg f^{(i)}(\bar{z}^{(i)})} \to X_{1,\bar{z}^{(j)}}^{f^{(j)}(\bar{z}^{(j)})}$ is *not* in the transitive closure of G_Φ. Otherwise, if such an edge is present, when we force $X_{1,\mathtt{M}(\bar{v}_i)}$ to be $\neg f^{(i)}(\mathtt{M}(\bar{v}_i))$ and $X_{1,\mathtt{M}(\bar{v}_j)}$ to be $\neg f^{(j)}(\mathtt{M}(\bar{v}_j))$, we will find a bad cycle and there will be no solutions.

We present the algorithm to compute N_C formally as Algorithm 2. We first split C into two sets C_1 and C_2 that contain the literals associated with y_1 and y_2, respectively.

In Line 8 we force the assignment by adding into T the edge $X_{1,\mathtt{M}(\bar{v}_i)}^{f^{(i)}(\mathtt{M}(\bar{v}_i))} \to X_{1,\mathtt{M}(\bar{v}_i)}^{\neg f^{(i)}(\mathtt{M}(\bar{v}_i))}$, for every $1 \leq i \leq t$. In Line 9 we run the command `reach` from `ABC` to check whether there is a cycle containing contradicting literals. If there is such a cycle, the assignment $\mathtt{M}$ is blocked in Line 11. In Line 13 we extract a Skolem function using the technique from (Fung et al. 2024). In Line 14 we count the number of solutions for the 1-DQBF instance after substituting y_1 with the new Skolem function f. Finally, in Line 16 we update the formula A by conjoining it with the conjunction in Eq. (3).

6 Experiments

We implemented the algorithm from the previous section in a tool called `sharp2DQR`. Formulas such as R, G_Φ, φ_{tr}, $\mathcal{S}_1$ and $\mathcal{S}_2$ are represented as BDDs using `cudd` (Somenzi 2009). This offers several advantages. For example, the formula φ_{tr} can be computed using BDD-based reachability as implemented in ABC's `reach` command (Brayton and Mishchenko 2010). The number of support/non-support variables can also be computed easily by constructing the

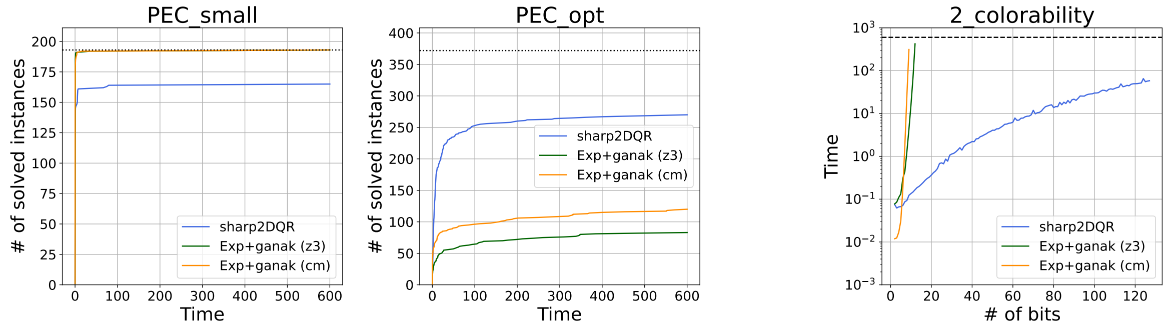

Figure 1: The two figures on the left shows the performance of the solvers on the PEC instances, the horizontal axis corresponds to the running time (s), and the vertical axis to the number of solved instances. The figure on the right shows the performance of the solvers on the 2-colorability instances, the horizontal axis corresponds to the number of bits of the graph in the instance, and the vertical axis to the running time (s).

BDD for $\mathcal{S}_1$ and $\mathcal{S}_2$ from $\neg\varphi$ with existential quantification.

To evaluate our model counter, we generated a diverse family of benchmarks which we divide into three batches of instances.

- PEC_opt: These are instances with dependency set size of 10 to 50. They are generated in a similar manner as in (Fung et al. 2024). There are 370 instances in this batch. One third of these have 0 non-support variables.
- PEC_small: These are instances generated from the ISCAS89 instances with dependency set size of 3 to 10 variables. There are 192 instances in this batch.
- 2_colorability: These are the 2 colorability instances as in (Fung et al. 2024). These are succinctly represented graphs with 2 to 127 bits, each of them contains exactly two Skolem functions.

For PEC_opt and PEC_small, the number of Skolem functions ranges from 1 to more than $2^{2^{64}}$ and the number of connected components ranges from 1 to more than 1600.

We evaluated the performance of `sharp2DQR` against `ganak` (Sharma et al. 2019), which is used to count the number of models of the expansion as defined in Eq. (2). The expansion is computed via `cryptominisat5` (Soos, Nohl, and Castelluccia 2009) or `z3` (De Moura and Bjørner 2008) as follows: First, we obtain a model $\mathtt{M}$ of $\neg\varphi$. Then, we generate the corresponding clauses in the expansion. Then, we add a blocking clause $\wedge_{x\in\bar{z}_1\cup\bar{z}_2\cup\{y_1,y_2\}} x \neq \mathtt{M}[x]$ to $\neg\varphi$ to prevent duplicated solutions and repeat until the formula becomes unsatisfiable. This method is called `Exp+ganak`.

The experiments were conducted on Ubuntu 22.04.4 LTS with 48 GB of 2400MHz DDR4 memory and an i5-13400 CPU. Each solver had 600 seconds to solve each instance.

Figure 1 shows that `sharp2DQR` fell short on PEC_small instances, but it is significantly better than `Exp+ganak` on PEC_opt instances. This is because the dependency set size is larger on PEC_opt instances, and since the expansion size is exponential to the dependency set size, our method, without expanding, performs better on larger instances. Most of the time spent by `Exp+ganak` is used on computing the expansion, and on many instances `ganak` finishes counting quite quickly after the expansion. `sharp2DQR` does not work well on small instances because sometimes BDD operations takes too long, and on the PEC_small instances, the number of unsatisfying models is small enough that enumeration is not too much of a problem. We also notice that for `Exp+ganak`, the performance of using `cryptominisat5` and `z3` is similar, but `cryptominisat5` is better on the PEC_opt instances.

For 2_colorability, `Exp+ganak` was unable to solve any instances larger than 12 bits, while `sharp2DQR` successfully solved instances up to 127 bits. This is due to the fact that the number of clauses in the expansion is $\Theta(2^n)$ for an n-bit graph, making full expansion very expensive.

More experimental results and analysis can be found in the supplementary material. We also compared both `Exp+ganak` and `sharp2DQR` against the latest FOMC tool `WFOMC` (Wang 2025), where we encode counting the number of 2-coloring and independent sets on some specific graphs. In all instances, `WFOMC` can only handle model sizes of up to 4, far lower than what `sharp2DQR` can handle, which in some instances is 2^{127}. However, in the independent set counting instances, `Exp+ganak` performs better than `sharp2DQR`.

7 Concluding Remarks

We established that #2-DQBF is as hard as general #DQBF. Specifically, we proved that it is #EXP-complete by leveraging the connections between k-DQBF and k-SAT (Fung and Tan 2023) and the technique in (Bannach et al. 2025).

On the experimental front, we introduced a novel algorithm for #2-DQBF using BDD-based symbolic reachability. As a baseline, we also implemented an approach that relies on universal expansion followed by propositional model counting. While our algorithm scaled better with larger dependency sets, the expansion-based method works for general DQBF and may be worth exploring further.

To the best of our knowledge, this is the first paper investigating model counting for DQBF, and there are many avenues for future research. One natural next step is to generalize our algorithm to handle 3-DQBF.

Acknowledgement

This paper contains the supplementary material of the conference paper (Fung et al. 2026). We thank the reviewers of AAAI 2026 for their detailed and constructive comments on the initial version.

We would like to acknowledge the generous support of the Royal Society International Exchange Grant no. R3\233183, the National Science and Technology Council of Taiwan grant no. 111-2923-E-002-013-MY3, and the NTU Center of Data Intelligence: Technologies, Applications, and Systems grant no. NTU-113L900903.

References

Balabanov, V.; Chiang, H. K.; and Jiang, J. R. 2014. Henkin quantifiers and Boolean formulae: A certification perspective of DQBF. *Theor. Comput. Sci.*, 523: 86–100.

Balabanov, V.; and Jiang, J. R. 2015. Reducing Satisfiability and Reachability to DQBF. In *QBF Workshop*.

Bannach, M.; Demaine, E. D.; Gomez, T.; and Hecher, M. 2025. #P is Sandwiched by One and Two #2DNF Calls: Is Subtraction Stronger Than We Thought? In *LICS*.

Beame, P.; den Broeck, G. V.; Gribkoff, E.; and Suciu, D. 2015. Symmetric Weighted First-Order Model Counting. In *PODS*.

Biere, A.; Fleury, M.; Froleyks, N.; and Heule, M. J. H. 2023. The SAT Museum. In Järvisalo, M.; and Berre, D. L., eds., *SAT*.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2009. *Handbook of Satisfiability*. IOS Press.

Bloem, R.; Könighofer, R.; and Seidl, M. 2014. SAT-Based Synthesis Methods for Safety Specs. In *VMCAI*.

Börger, E.; Grädel, E.; and Gurevich, Y. 1997. *The Classical Decision Problem*. Springer.

Bradley, A. 2011. SAT-Based Model Checking without Unrolling. In *VMCAI*.

Bradley, A.; and Manna, Z. 2007. Checking Safety by Inductive Generalization of Counterexamples to Induction. In *FMCAD*.

Brayton, R.; and Mishchenko, A. 2010. ABC: An Academic Industrial-Strength Verification Tool. In *CAV*.

Capelli, F.; Lagniez, J.; Plank, A.; and Seidl, M. 2024. A Top-Down Tree Model Counter for Quantified Boolean Formulas. In *IJCAI*.

Chatterjee, K.; Henzinger, T.; Otop, J.; and Pavlogiannis, A. 2013. Distributed Synthesis for LTL Fragments. In *FMCAD*.

Chen, F.-H.; Huang, S.-C.; Lu, Y.-C.; and Tan, T. 2022. Reducing NEXP-complete problems to DQBF. In *FMCAD*.

Cheng, C.; Fung, L.-H.; Jiang, J.-H. R.; Slivovsky, F.; and Tan, T. 2025. Fine-Grained Complexity Analysis of Dependency Quantified Boolean Formulas. In *SAT*.

Dalvi, N.; and Suciu, D. 2004. Efficient Query Evaluation on Probabilistic Databases. In *VLDB*.

De Moura, L.; and Bjørner, N. 2008. Z3: An efficient SMT solver. In *TACAS*.

Eén, N.; Mishchenko, A.; and Brayton, R. 2011. Efficient Implementation of Property Directed Reachability. In *FMCAD*.

Fichte, J. K.; Berre, D. L.; Hecher, M.; and Szeider, S. 2023. The Silent (R)evolution of SAT. *Commun. ACM*, 66(6): 64–72.

Fröhlich, A.; Kovásznai, G.; Biere, A.; and Veith, H. 2014. iDQ: Instantiation-Based DQBF Solving. In *Pragmatics of SAT Workshop (POS)*.

Fung, L.; and Tan, T. 2023. On the Complexity of k-DQBF. In *SAT*.

Fung, L.-H.; Cheng, C.; Fan, Y.-W.; Tan, T.; and Jiang, J.-H. R. 2024. 2-DQBF Solving and Certification via Property-Directed Reachability Analysis. In *FMCAD*.

Fung, L.-H.; Cheng, C.; Jiang, J.-H. R.; Slivovsky, F.; and Tan, T. 2026. Model Counting for Dependency Quantified Boolean Formulas. In *AAAI*.

Galperin, H.; and Wigderson, A. 1983. Succinct Representations of Graphs. *Inf. Control.*, 56(3): 183–198.

Ge-Ernst, A.; Scholl, C.; Síc, J.; and Wimmer, R. 2022. Solving Dependency Quantified Boolean Formulas using Quantifier Localization. *Theor. Comput. Sci.*, 925: 1–24.

Gitina, K.; Reimer, S.; Sauer, M.; Wimmer, R.; Scholl, C.; and Becker, B. 2013a. Equivalence checking of partial designs using dependency quantified Boolean formulae. In *ICCD*.

Gitina, K.; Reimer, S.; Sauer, M.; Wimmer, R.; Scholl, C.; and Becker, B. 2013b. Equivalence checking of partial designs using dependency quantified Boolean formulae. In *ICCD*.

Gitina, K.; Wimmer, R.; Reimer, S.; Sauer, M.; Scholl, C.; and Becker, B. 2015. Solving DQBF through quantifier elimination. In *DATE*.

Golia, P.; Roy, S.; and Meel, K. S. 2021. Program Synthesis as Dependency Quantified Formula Modulo Theory. In *IJCAI*.

Golia, P.; Roy, S.; and Meel, K. S. 2023. Synthesis with Explicit Dependencies. In *DATE*.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2021. Model Counting. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 993–1014. IOS Press.

Jiang, J. R. 2009. Quantifier Elimination via Functional Composition. In *CAV*.

Kuehlmann, A.; Paruthi, V.; Krohm, F.; and Ganai, M. 2002. Robust Boolean reasoning for equivalence checking and functional property verification. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 21(12): 1377–1394.

Ladner, R. 1989. Polynomial Space Counting Problems. *SIAM J. Comput.*, 18(6): 1087–1097.

Lewis, H. 1980. Complexity Results for Classes of Quantificational Formulas. *J. Comput. Syst. Sci.*, 21(3): 317–353.

Papadimitriou, C.; and Yannakakis, M. 1986. A Note on Succinct Representations of Graphs. *Inf. Control.*, 71(3): 181–185.

Peterson, G.; and Reif, J. 1979. Multiple-Person Alternation. In *FOCS*.

Plank, A.; Möhle, S.; and Seidl, M. 2024. Counting QBF solutions at level two. *Constraints*, 29(1-2): 22–39.

Pulina, L.; and Seidl, M. 2019. The 2016 and 2017 QBF solvers evaluations (QBFEVAL'16 and QBFEVAL'17). *Artif. Intell.*, 274: 224–248.

Reichl, F.; and Slivovsky, F. 2022. Pedant: A Certifying DQBF Solver. In *SAT*.

Reichl, F.; Slivovsky, F.; and Szeider, S. 2021. Certified DQBF Solving by Definition Extraction. In *SAT*.

Richardson, M.; and Domingos, P. M. 2006. Markov logic networks. *Mach. Learn.*, 62(1-2): 107–136.

Scholl, C.; and Becker, B. 2001. Checking Equivalence for Partial Implementations. In *DAC*.

Sharma, S.; Roy, S.; Soos, M.; and Meel, K. S. 2019. GANAK: A Scalable Probabilistic Exact Model Counter. In *IJCAI*.

Shaw, A.; Juba, B.; and Meel, K. S. 2024. An approximate skolem function counter. In *AAAI*.

Síc, J.; and Strejcek, J. 2021. DQBDD: An Efficient BDD-Based DQBF Solver. In *SAT*.

Skyum, S.; and Valiant, L. 1985. A Complexity Theory Based on Boolean Algebra. *J. ACM*, 32(2): 484–502.

Somenzi, F. 2009. CUDD: CU decision diagram package release 2.4. 2. *University of Colorado at Boulder*.

Soos, M.; Nohl, K.; and Castelluccia, C. 2009. Extending SAT Solvers to Cryptographic Problems. In Kullmann, O., ed., *SAT*.

Tentrup, L.; and Rabe, M. 2019. Clausal Abstraction for DQBF. In *SAT*.

Tóth, J.; and Kuzelka, O. 2024. Complexity of Weighted First-Order Model Counting in the Two-Variable Fragment with Counting Quantifiers: A Bound to Beat. In *KR*.

Trakhtenbrot, B. 1950. The impossibility of an algorithm for the decidability problem on finite classes. In *D. Akad. Nauk USSR, 70(1)*, 569–572.

Valiant, L. 1979a. The Complexity of Computing the Permanent. *Theor. Comput. Sci.*, 8: 189–201.

Valiant, L. 1979b. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.*, 8(3): 410–421.

van Bremen, T.; and Kuzelka, O. 2023. Lifted inference with tree axioms. *Artif. Intell.*, 324: 103997.

Vardi, M. Y. 1982. The Complexity of Relational Query Languages (Extended Abstract). In *STOC*.

Wang, Y. 2025. https://github.com/yuanhong-wang/WFOMC. Accessed: 2025-07-28.

Wimmer, R.; Karrenbauer, A.; Becker, R.; Scholl, C.; and Becker, B. 2017. From DQBF to QBF by Dependency Elimination. In *SAT*.

More detailed proofs and experimental statistics
"Model Counting for Dependency Quantified Boolean Formulas"

In this supplementary material all logarithms have base 2.

A Parsimonious reduction from general DQBF to uniform DQBF

Recall that a k-DQBF Φ is uniform, if for every Skolem functions $(f_1, \ldots, f_k) \models \Phi$, all the functions $f_1, \ldots, f_k$ must be the same function. Here we will show the polynomial time parsimonious reduction from general DQBF to uniform DQBF. The intuition is that we can combine k functions $f_1, \ldots, f_k$ into one function f. Suppose $f_i : \mathbb{B}^{n_i} \to \mathbb{B}$, for each $1 \leq i \leq k$. We can combine it into one function $f : \mathbb{B}^{\log k} \times \mathbb{B}^n \to \mathbb{B}$, where n is the maximal among $n_1, \ldots, n_k$. The first $\log k$ bits can be used as the index of the function.

We now give the formal construction. We fix an arbitrary k-DQBF:

$$\Psi \ := \ \forall \bar{x}\, \exists y_1(\bar{z}_1) \cdots \ \exists y_k(\bar{z}_k) \ \ \psi$$

where $\bar{x} = (x_1, \ldots, x_n)$, each $\bar{z}_i \subseteq \bar{x}$.

We construct the following uniform k-DQBF:

$$\Psi' = \forall \bar{x}_1, \ldots, \forall \bar{x}_k, \forall \bar{u}_1, \ldots, \forall \bar{u}_k, \exists y_1(\bar{x}_1 \cup \bar{u}_1), \ldots, \exists y_k(\bar{x}_k \cup \bar{u}_k).\, \psi' \,,$$

where each $\bar{x}_i$ is a copy of $\bar{x}$, each $\bar{u}_i$ is of $\lceil \log k \rceil$ bits encoding a number in $[2^{\lceil \log k \rceil}]$, and ψ' is the conjunction of the following:

- $((\bar{x}_i = \bar{x}_j) \wedge (\bar{u}_i = \bar{u}_j)) \to (y_i \leftrightarrow y_j)$ for every $1 \leq i < j \leq k$,
- $((\bar{x}_1|_{\bar{z}_i} = \bar{x}_2|_{\bar{z}_i}) \wedge (\bar{u}_1 = \bar{u}_2 = i)) \to (y_1 \leftrightarrow y_2)$ for every $i \in [k]$,
- $\left(\bigwedge_{i \in [k]} \bar{u}_i = i\right) \to \psi$, and
- $(\bar{u}_i = j) \to y_i$ for every $i \in [k]$ and $k + 1 \leq j \leq 2^{\lceil \log k \rceil}$.

The first item ensures that the DQBF is uniform. The second ensures that the dependencies of the original existential variables are respected. The third encodes the original formula. The last enforces that there is a unique Skolem function for the unused indices in order to preserve the number of Skolem functions.

For every Skolem function $(f_1, \ldots, f_k)$, consider the function:

$$f(i, \bar{x}) = \begin{cases} f_i(\bar{x}|_{\bar{z}}) & \text{if } i \leq k \\ 1 & \text{otherwise} \end{cases}$$

which forms a Skolem function for Ψ' by copying itself k times. For any Skolem function of Ψ', it must be k copies of a function f. Consider the function:

$$f_i(\bar{z}_i) = f(i, \mathrm{Ext}(\bar{z}_i, \bar{x}))$$

where $\mathrm{Ext}(\bar{z}_i, \bar{x})$ lifts $\bar{z}_i$ to $\bar{x}$ by substituting the variables not in $\bar{z}_i$ to 0. Then $(f_1, \ldots, f_k)$ is a Skolem function for Ψ. It is clear that this is a bijection.

B Proof of Lemma 5: An explicit reduction from #DQBF to #2-DQBF

In this section we present the proof of Lemma 5:

> *There is a reduction that transforms a DQBF Ψ to two 2-DQBF Φ_1 and Φ_2 such that $\#\Psi = \#\Phi_1 - \#\Phi_2$. The reduction runs in time $O(k^2|\psi|)$, where k is the number of existential variables in Ψ and ψ is the matrix of Ψ.*

We will first briefly review the reduction in (Bannach et al. 2025). For the sake of presentation, we will present only the simplified version, which is already sufficient for our purpose. It should also be clear from our presentation that the reduction is indeed projection as required in the proof of Lemma 4. We then show how to lift the reduction to the DQBF setting. We will first show the reduction to a slightly generalized version of 2-DQBF that we call *extended 2-DQBF*. Then, we present a parsimonious reduction from extended 2-DQBF to 2-DQBF.

The reduction from $\#$SAT to $\#$2-SAT:

Given a CNF φ over the variables $\bar{x} = (x_1, \ldots, x_n)$:

$$\varphi := \bigwedge_{i=1}^{m} C_i, \quad \text{where each } C_i = \ell_{i,1} \vee \cdots \vee \ell_{i,k_i}$$

We construct a 2-CNF forula ϕ over the variables $\bar{x} \cup \{c_1, \ldots, c_m\} \cup \{p_i, o1_i, o2_i, e1_i, e2_i\}_{i=0,\ldots,m}$. The intended meaning of each variable is as follows.

- Variable c_i is the indicator whether clause C_i is falsified: It is true only if clause C_i is falsified.
- Variable p_i denotes the parity of the number of clauses falsified up to clause i.
- Variables $o1_i$, $o2_i$, $e1_i$, and $e2_i$ are used to represent different cases of the parity of the number of clauses falsified up to clause i.

The formula ϕ is constructed to reflect the intended meaning of the variables. Formally, it is the conjunction of the following clauses:

(R1) For each $i = 1, \ldots, m$, and every literal $\ell_{i,j}$, we have the clause:

$$c_i \to \neg \ell_{i,j} \tag{4}$$

That is, if clause C_i is falsified, then all its literals must be falsified.

(R2) For each $i = 1, \ldots, m$, we have the following four groups of clauses:

- The number of clauses falsified up to clause i is odd because clause i is falsified.

$$o1_i \to \neg p_{i-1} \qquad o1_i \to c_i \qquad o1_i \to p_i \tag{5}$$

- The number of clauses falsified up to clause i is odd because clause i is not falsified.

$$o2_i \to p_{i-1} \qquad o2_i \to \neg c_i \qquad o2_i \to p_i \tag{6}$$

- The number of clauses falsified up to clause i is even because clause i is falsified.

$$e1_i \to p_{i-1} \qquad e1_i \to c_i \qquad e1_i \to \neg p_i \tag{7}$$

- The number of clauses falsified up to clause i is even because clause i is not falsified.

$$e2_i \to \neg p_{i-1} \qquad e2_i \to \neg c_i \qquad e2_i \to \neg p_i \tag{8}$$

(R3) Finally we have the clause:

$$\neg p_0 \tag{9}$$

Intuitively, this means that initially there is zero clause falsified.

In (Bannach et al. 2025), it is shown that:

$$\#(\varphi) \;=\; \#(\phi \wedge \neg p_m) - \#(\phi \wedge p_m)$$

The proof is essentially the inclusion-exclusion principle, where we count the number of falsifying assignments.

The reduction from $\#$DQBF to $\#2$-DQBF:

We will show how to reduce DQBF to *extended 2-DQBF* defined as follows.

Definition 12. *A DQBF is an* extended 2-DQBF *if its matrix is a conjunction $\bigwedge_i \varphi_i$, where each φ_i uses at most two existential variables.*

In the following we will show the desired reduction from DQBF to extended 2-DQBF which suffices for our purpose thanks to the following lemma.

Lemma 13. *There is a parsimonious polynomial time reduction that transforms extended 2-DQBF instances to 2-DQBF instances.*

Proof. The proof is rather similar to Appendix A, where we combine several functions into one function.

Let the given extended 2-DQBF be:

$$\forall \bar{x} \exists y_1(\bar{z}_1) \ldots \exists y_k(\bar{z}_k)\ \varphi, \quad \text{where } \varphi := \bigwedge_{1 \le i < j \le m} \varphi(\bar{x}, \bar{y}_i, \bar{y}_j).$$

Consider the 2-DQBF:

$$\forall \bar{x} \forall \bar{x}' \forall \bar{i} \forall \bar{i}' \exists y(\bar{x}, \bar{i}) \exists y'(\bar{x}', \bar{i}').\ \psi$$

where

$$\psi' = \bigwedge_{k=1}^{m} ((\bar{i} = \bar{i}' = k \wedge \bar{z}_k = \bar{z}'_k) \to y = y') \ \wedge \ \left(\bar{x} \neq \bar{x}' \vee \bigwedge_{1 \le k < \ell \le m} ((\bar{i} = k \wedge \bar{i}' = \ell) \to \varphi_{k,\ell}[y_k/y, y_\ell/y']) \right)$$

Intuitively, the function y represents y_k when $i = k$. When $i = i' = k$ and k is between 1 and m, the formula $(\bar{i} = \bar{i}' = k \wedge \bar{z}_k = \bar{z}'_k) \to y = y'$ forces y and y' to be the same function, and is independent of variables outside $\bar{z}_k$. Since each $\varphi_{k,\ell}$ only contains two existential variables, y_k and y_ℓ, $(\bar{i} = k \wedge \bar{i}' = \ell) \to \varphi_{k,\ell}[y_k/y, y_\ell/y']$ ensures that y_k and y_ℓ represented by y satisfies the $\varphi_{k,\ell}$ when $\bar{x} = \bar{x}'$.

□

We now give the reduction as required in Lemma 5. Let the input k-DQBF be:

$$\Phi := \forall x_1 \cdots x_n \exists y_1(\bar{z}_1) \cdots y_k(\bar{z}_k)\ \varphi.$$

Let $\Lambda = \{0,1\}^n \times \{0,1\}^k$ and $\Lambda^* = \Lambda \cup \{-1\}$. We will view elements in Λ as integers between 0 and $2^{n+k} - 1$ (inclusive), and elements in Λ^* as integers between -1 and $2^{n+k} - 1$ (inclusive). By encoding -1 with an additional bit and abuse of notation, we may treat Λ^* as a subset of $\mathbb{B}^{n+k+1}$.

Consider the following DQBF:

$$\Psi := \forall (\bar{x}, \bar{y}) \in \Lambda\ \forall \bar{t} \in \Lambda^*\ \forall \bar{t}' \in \Lambda^*\ \exists f_1(\bar{z}_1) \cdots \exists f_k(\bar{z}_k)\ \exists c(\bar{x}, \bar{y})\ \exists p(\bar{t}')\ \exists o1(\bar{t})\ \exists o2(\bar{t})\ \exists e1(\bar{t})\ \exists e2(\bar{t})\ \ \psi$$

where $|\bar{x}| = n$, $|\bar{y}| = k$, $|\bar{t}| = |\bar{t}'| = n + k + 1$ and the matrix ψ is the conjunction of the following

(S1)

- $\neg\varphi \to (y_i \to (c \to f_i))$ for each $i \in \{1, \ldots, k\}$
- $\neg\varphi \to (\neg y_i \to (c \to \neg f_i))$ for each $i \in \{1, \ldots, k\}$
- $\varphi \to \neg c$

(S2)

- $(t \neq -1 \wedge t' = t - 1) \to (o1 \to \neg p)$
- $(t \neq -1 \wedge t = (\bar{x}, \bar{y})) \to (o1 \to c)$
- $(t \neq -1 \wedge t' = t) \to (o1 \to p)$

- $(t \neq -1 \wedge t' = t - 1) \to (o2 \to p)$
- $(t \neq -1 \wedge t = (\bar{x}, \bar{y})) \to (o2 \to \neg c)$
- $(t \neq -1 \wedge t' = t) \to (o2 \to p)$

- $(t \neq -1 \wedge t' = t - 1) \to (e1 \to p)$
- $(t \neq -1 \wedge t = (\bar{x}, \bar{y})) \to (e1 \to c)$
- $(t \neq -1 \wedge t' = t) \to (e1 \to \neg p)$

- $(t \neq -1 \wedge t' = t - 1) \to (e2 \to \neg p)$
- $(t \neq -1 \wedge t = (\bar{x}, \bar{y})) \to (e2 \to \neg c)$
- $(t \neq -1 \wedge t' = t) \to (e2 \to \neg p)$

(S3)

- $(t' = -1) \to \neg p$

The formulas in (S1) correspond to the clauses in (R1), (S2) to the clauses in (R2) and (S3) to the clause in (R3).

Consider the following two 2-DQBF Ψ_1 and Ψ_2.

- Ψ_1 is obtained from Ψ by conjuncting its matrix with $(t' = 2^{n+3} - 1) \to \neg p$.
- Ψ_2 is obtained from Ψ by conjuncting its matrix with $(t' = 2^{n+3} - 1) \to p$.

It can be shown that:

$$\#\Phi = \#\Psi_1 - \#\Psi_2.$$

The proof is similar to (Bannach et al. 2025). We observe that the expansion of Ψ_1 and Ψ_2 are essentially the same as in the previous subsection. Note also that both Ψ_1 and Ψ_2 are extended 2-DQBF.

C Missing details in the proof of Theorem 8

We first recall the reduction in Theorem 8. Suppose we are given a uniform 2-DQBF:

$$\Phi := \forall \bar{x} \exists y_1(I_1) \exists y_2(I_2) \phi,$$

where $\bar{x} = (x_1, \ldots, x_n)$, $I = \{i_1, \ldots, i_m\}$ and $J = \{j_1, \ldots, j_m\}$. We construct the following FO sentence Ψ using only one predicate symbol S with arity m:

$$\Psi \ := \ \exists u_0 \exists u_1 \forall v_1 \cdots \forall v_n \ (u_0 \neq u_1) \wedge \psi$$

where ψ is the formula obtained from ϕ by replacing:

- each x_i in ϕ with $v_i = u_1$ for every $1 \leq i \leq n$;
- y_1 and y_2 with $S(v_{i_1}, \ldots, v_{i_m})$ and $S(v_{j_1}, \ldots, v_{j_m})$, resp.

The rest of this appendix is devoted to the proof that $\#\Phi$ is half the number of models of Ψ with domain $\{1, 2\}$. To avoid being repetitive, in this section, the domain of first-order structures is assumed to be $\{1, 2\}$.

We first show that swapping the roles of 1 and 2 in a structure does not effect the satisfiability of Ψ. Suppose $\mathcal{A} \models \Psi$. Let $\mathcal{A}^*$ be the structure obtained by swapping the roles of 1 and 2 in $\mathcal{A}$. We claim that $\mathcal{A} \models \Psi$ iff $\mathcal{A}^* \models \Psi$. Indeed, if $\mathcal{A} \models \Psi$, by definition, there is an assignment to u_0, u_1 with the elements in $\{1, 2\}$. Due to $u_0 \neq u_1$, the assignments must be different. Suppose u_0 is assigned with 1 and u_2 is assigned with 2. Then, $\mathcal{A}^*$ also satisfies Ψ by assigning u_0 with 2 and u_1 with 1. That $\mathcal{A}^* \models \Psi$ implies $\mathcal{A} \models \Psi$ is analogous.

Next, we show that a Boolean function $f : \mathbb{B}^m \to \mathbb{B}$ corresponds uniquely to two structures $\mathcal{A}_{1,f}$ and $\mathcal{A}_{2,f}$. We use the following notation. For each $\bar{a} = (a_1, \ldots, a_m) \in \mathbb{B}^m$, define $\widetilde{\bar{a}} = (\widetilde{a}_1, \ldots, \widetilde{a}_m) \in \{1, 2\}^m$, where for each $i \in \{1, \ldots, m\}$:

$$\widetilde{a}_i \ := \ \begin{cases} 1 & \text{if } a_i = \top \\ 2 & \text{if } a_i = \bot \end{cases}$$

For each function $f : \mathbb{B}^m \to \mathbb{B}$, we define the structures $\mathcal{A}_{1,f}$ where for each $\bar{a} \in \mathbb{B}^m$:

$$f(\bar{a}) = \top \ \text{ if and only if } \ \widetilde{\bar{a}} \in S$$

The structure $\mathcal{A}_{2,f}$ is obtained by swapping the roles of 1 and 2 in $\mathcal{A}_{1,f}$. It is routine to verify that $(f, f) \models \Phi$ iff both $\mathcal{A}_{1,f}$ and $\mathcal{A}_{2,f}$ satisfy Ψ. This implies that $\#\Phi$ is half the number of models of Ψ.

D Proof of Corollary 9

Recall Corollary 9:

> *There is a fragment $\mathcal{L}$ of FO of which the satisfiability problem is in* PSPACE*, but its corresponding FOMC is* #EXP*-complete even when the domain size is restricted to* 2*.*

The desired logic $\mathcal{L}$ is a subclass of Bernays-Schoenfinkel-Ramsey class with relation symbol S (with varying arity) and arbitrary number of unary relation symbols. It contains sentences of the form:

$$\Psi \ := \ \exists u_1 \cdots \exists u_k \forall v_1 \cdots \forall v_n \ \psi \tag{10}$$

where the number of atoms using the relation S is limited to 2. This class already captures the sentence used in the proof of hardness (where the equality predicate $u_0 \neq u_1$ is replaced with $U(u_1) \wedge \neg U(u_0)$). Therefore its model counting is #EXP-complete. What is left is to show that this logic is decidable in polynomial space.

Let the input sentence be Ψ as in Eq. (10). We will use the well known fact that if Ψ is satisfiable, then it is satisfiable by a model of size at most k (Börger, Grädel, and Gurevich 1997). Let $U_1, \ldots, U_t$ be the unary predicates used in Ψ. Let the two atoms using the relation symbol S be $S(v_{i_1}, \ldots, v_{i_m})$ and $S(v_{j_1}, \ldots, v_{j_m})$. The (non-deterministic) polynomial space algorithm for deciding the satisfiability of Ψ works as follows.

1. Guess the size of the model s between 1 and k.
   The domain of the model is assumed to be a subset of $\mathbb{B}^{\log s}$.
2. For each $i \in \{1, \ldots, k\}$, guess the element a_i to be assigned to u_i.
3. For each $i \in \{1, \ldots, t\}$, guess the unary predicate $U_i := \{b_{i,1}, \ldots, b_{i,p_i}\}$.
4. Construct the following uniform 2-DQBF:

$$\Phi \ := \forall \bar{x}_1 \cdots \forall \bar{x}_n \exists y_1(\bar{x}_{i_1}, \ldots, \bar{x}_{i_m}) \exists y_2(\bar{x}_{j_1}, \ldots, \bar{x}_{j_m})$$

$$\Big((\bar{x}_{i_1}, \ldots, \bar{x}_{i_m}) = (\bar{x}_{j_1}, \ldots, \bar{x}_{j_m}) \to y_1 = y_2 \Big) \ \wedge \ \left(\left(\bigwedge_{1 \leq i \leq n} \mathrm{num}(\bar{x}_i) \leq s - 1 \right) \ \to \ \phi \right)$$

where:

- $|\bar{x}_1| = \cdots = |\bar{x}_m| = |\bar{w}_1| = \cdots = |\bar{w}_m| = \log s$.
- $\mathrm{num}(\bar{x}_i) \le s-1$ is the formula stating that the number represented by $\bar{x}_i, \bar{w}_i$ are less than or equal to $s-1$.
- ϕ is the formula obtained from ψ by performing the following.
  - Replace each atom $v_j = u_{j'}$ with $\bar{x}_j = a_{j'}$.
  - Replace each atom $v_j = v_{j'}$ with $\bar{x}_j = \bar{x}_{j'}$.
  - Replace each atom $U_i(u_j)$ with $\top$ or $\bot$ depending on whether a_j is in the guessed set U_i.
  - Replace each atom $U_i(v_j)$ with the disjunction $(\bar{x}_j = b_{i,1}) \vee \cdots \vee (\bar{x}_j = b_{i,p_i})$.
  - Replace each atom $S(v_{i_1}, \ldots, v_{i_m})$ with y_1.
  - Replace each atom $S(v_{j_1}, \ldots, v_{j_m})$ with y_2.

It is routine to verify that Ψ is satisfiable if and only if there is a guess for the number s, the set $\{b_{i,1}, \ldots, b_{i,p_i}\}$ for each $1 \le i \le t$ and the element a_i for each $1 \le i \le k$ such that Φ is satisfiable. Since Φ is 2-DQBF whose satisfiability can be checked in polynomial space (Fung and Tan 2023), the whole algorithm runs in polynomial space.

E Proof of Lemma 10

We recall Lemma 10:

Let $\mathcal{S}_1 := \{\bar{c} : \neg\varphi[\bar{z}_1/\bar{c}]$ is satisfiable$\}$ and $\mathcal{S}_2 := \{\bar{c} : \neg\varphi[\bar{z}_2/\bar{c}]$ is satisfiable$\}$. The set of support variables in $\exp(\Phi)$ is $\{X_{1,\bar{c}} : \bar{c} \in \mathcal{S}_1\} \cup \{X_{2,\bar{c}} : \bar{c} \in \mathcal{S}_2\}$. Moreover, the number of support and non-support variables is $|\mathcal{S}_1| + |\mathcal{S}_2|$ and $(2^{|\bar{z}_1|} - |\mathcal{S}_1|) + (2^{|\bar{z}_2|} - |\mathcal{S}_2|)$, respectively.

By definition, $X_{1,\bar{c}}$ is a support variable iff there is a clause $C_{\bar{a},\bar{b}}$ that contains it, which is equivalent to $\varphi[(\bar{x},\bar{y})/(\bar{a},\bar{b})] = \bot$ and $\bar{c} = \bar{a}\big|_{\bar{x}\downarrow\bar{z}_1}$. By definition, $X_{1,\bar{c}}$ is a support variable iff $\bar{c} \in \mathcal{S}_1$. Similarly, $X_{2,\bar{c}}$ is a support variable iff $\bar{c} \in \mathcal{S}_2$. Thus, the number of support variables is $|\mathcal{S}_1| + |\mathcal{S}_2|$. Since there are $2^{|\bar{z}_1|}$ number of variables $X_{1,\bar{c}}$ and $2^{|\bar{z}_2|}$ number of variables $X_{2,\bar{c}}$, the number of non-support variables is $(2^{|\bar{z}_1|} - |\mathcal{S}_1|) + (2^{|\bar{z}_2|} - |\mathcal{S}_2|)$.

F Proof of Lemma 11

Recall Lemma 11:

Let $\Upsilon := \forall\bar{u}\exists y(\bar{v})\ \varphi$ be a satisfiable 1-DQBF.

- *The number of Skolem functions for Υ is 2^m, where $m = 2^{|\bar{v}|} - |\{\bar{c} : \neg\varphi[\bar{v}/\bar{c}]$ is satisfiable$\}|$.*
- *In particular, for a set $S \subseteq \mathbb{B}^{|\bar{v}|}$, the number of Skolem functions for Υ that differ on S is 2^m, where $m = |S| - |\{\bar{c} : \neg\varphi[\bar{v}/\bar{c}] \wedge (\bar{c} \in S)$ is satisfiable$\}|$.*

It suffices to prove the first bullet. The second bullet follows immediately. By definition, $\exp(\Upsilon)$ is a 1-CNF formula consisting of unit clauses. Since Υ is satisfiable, $\exp(\Upsilon)$ is satisfiable as well, and the unit clauses determine a unique satisfying assignment. This fixes the assignment of (support) variables in $\exp(\Upsilon)$, which is the the set of variables $X_{1,\bar{c}}$ where $\neg\varphi[\bar{v}/\bar{c}]$ is satisfiable. Since the non-support variables can be assigned arbitrarily and the total number of variables is $2^{|\bar{v}|}$, the lemma follows.

G Engineering Algorithms 1 and 2

This section contains some important details of our implementation of Algorithm 1.

Data structures. All important data structures such as R, G_Φ, $\mathcal{S}_1$ and $\mathcal{S}_2$ are stored as BDD. The BDD encoding of G_ϕ is necessary since explicitly constructing the graph G_Φ is infeasible.

We construct a BDD for the formula φ_{tr} that represents the transitive closure of G_Φ, i.e., $\varphi_{tr}(u,v) = 1$ iff u, v are both support variables and there is a path from u to v in the graph G_Φ. It can be obtained using BDD-based reachability, e.g., `reach` command from ABC (Brayton and Mishchenko 2010), on the transition system (I, T). Such BDD representation is useful in listing the Skolem function candidates.

Note also that by Lemma 10, we could have used the formula ϕ to represent the set R. However, this is not practical since the set R has to be updated continuously (in Line 9 in Algorithm 1). For this reason, we also use a BDD to represent R.

Due to the magnitude of the numbers N and N_C, we use the sparse integer representation where we only store the exponent in the binary representation. For example, the number 10100101 (in binary representation) is encoded as a list $\langle 0, 2, 5, 7\rangle$.

Computing the number of support/non-support variables. We construct the BDD for each $\mathcal{S}_1$ and $\mathcal{S}_2$ from $\neg\phi$ with existential abstraction, as defined in Lemma 10. Their cardinalities can be computed easily due to the BDD structure, e.g., `Cudd_CountMinterm` command from `cudd` package.

Picking an arbitrary variable $X_{i,\bar{c}}$ and the component C. To pick an arbitrary variable $X_{i,\bar{c}}$, we pick a satisfying assignment $\bar{c}$ from the BDD R. The component C is obtained by computing the closure of $X_{i,\bar{c}}$ in the implication graph. We remove C from R by intersecting R with the negation of C.

H Reduction from 2-DQBF to a symbolic reachability instance

In this section we briefly recall the reduction from 2-DQBF to a symbolic reachability instance as given in (Fung et al. 2024). For more details, please refer to (Fung et al. 2024). Given a 2-DQBF $\Phi := \forall \bar{x} \exists y_1(\bar{z}_1) \exists y_2(\bar{z}_2)\ \varphi$, the idea is to check if there is a cycle that contains both a literal and its negation in the implication graph of $\exp(\Phi)$. The formula $\neg\varphi$ is the succinct representation of the implication graph. For example, given two literals $L = X^{b_1}_{1,\bar{a}_1}$ and $L' = X^{b_2}_{2,\bar{a}_2}$, we can check if there is an edge from $X^{b_1}_{1,\bar{a}_1}$ to $X^{b_2}_{2,\bar{a}_2}$ by checking if $\neg\varphi \wedge \bar{z}_1 = \bar{a}_1 \wedge \bar{z}_2 = \bar{a}_2 \wedge y_1 = b_1 \wedge y_2 = \neg b_2$ is satisfiable.

We construct the transition system over the states (b, L, L_0) where $b \in \{0, 1\}$ and L, L_0 are literals of $\exp(\Phi)$. The initial condition is $b = 0 \wedge L = L_0$. The state (b, L, L_0) can transit to the state (b, L', L_0) if $E(L, L')$ and the state $(0, L, L_0)$ can transit to the state $(1, L, L_0)$ if $L = \neg L_0$. A state of the form $(1, \neg L_0, L_0)$ is reachable from the initial state if and only if the 2-DQBF Φ is unsatisfiable.

I More experimental results

More experimental results on PEC

Figure 2 shows pairwise comparisons between `sharp2DQR` and `Exp+ganak` on the PEC instance. Each point represents an instance. The axes in log scale represents the time needed for the corresponding solver. We can again see that for instances in PEC_opt, most points lie on the top left portion, i.e., `sharp2DQR` is better. However, for most instances in PEC_small, the points lie on the bottom right portion, i.e., `Exp+ganak` is better.

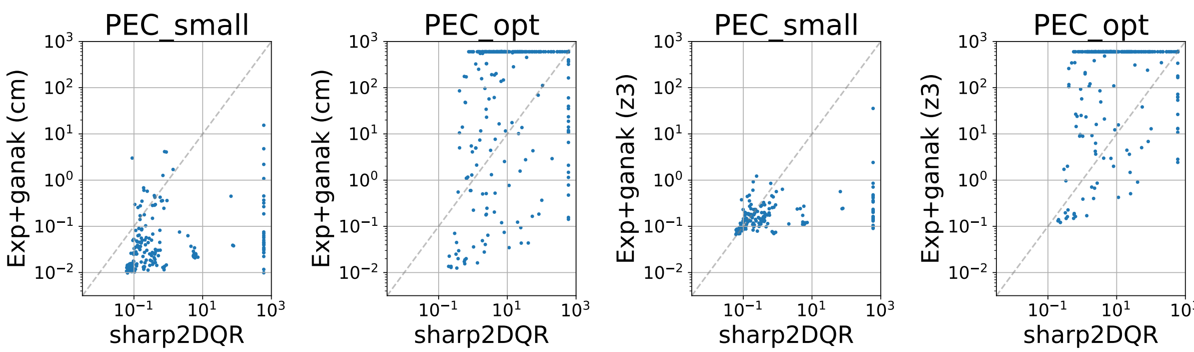

Figure 2: Pairwise comparison between `sharp2DQR` and `Exp+ganak`.

By separating the time for computing the expansion and counting the number of solutions as in Figure 4 for the 2-colorability instances, we notice that `z3` performs better on these instances. Additionally, for `z3`, the time spent on expansion was similar to the time used on counting the number of solutions with `ganak`, indicating that expansion is not the sole bottleneck.

2-colorability

Consider an n-bit graph with the edge circuit:

$$C_{n,k}(\bar{x}, \bar{x}') := \bigwedge_{i=1}^{k} x_i = x'_i \wedge x_{k+1} \neq x'_{k+1} \tag{11}$$

where $\bar{x} = (x_1, \ldots, x_n)$ and $\bar{x}' = (x'_1, \ldots, x'_n)$. The circuit represents a graph $G_{n,k}$ which is a union of 2^k complete bipartite graph and each component has size 2^{n-k-1}.

We consider the DQBF:

$$\text{TWO-COL}_{n,k} := \forall \bar{x} \forall \bar{x}' \exists y(\bar{x}) \exists y(\bar{x}').\ ((\bar{x} = \bar{x}') \to (y = y')) \wedge (E_{n,k} \to (y \neq y'))$$

When $k = 0$, these are the same instances as in (Fung et al. 2024). Each Skolem function of TWO-COL$_{n,k}$ corresponds to a 2-coloring of $G_{n,k}$. The number of Skolem functions for TWO-COL$_{n,k}$ is 2^k. Figure 3 shows the experimental results comparing `sharp2DQBR` and `Exp+ganak`. For 2-colorability instances `sharp2DQR` can again solve for instances larger than 15 bits while `Exp+ganak` can't. When k is small (up to 3), `sharp2DQR` can even handle instances up to 127 bits. However, for large k with $n \leq 15$, `Exp+ganak` outperforms `sharp2DQR`.

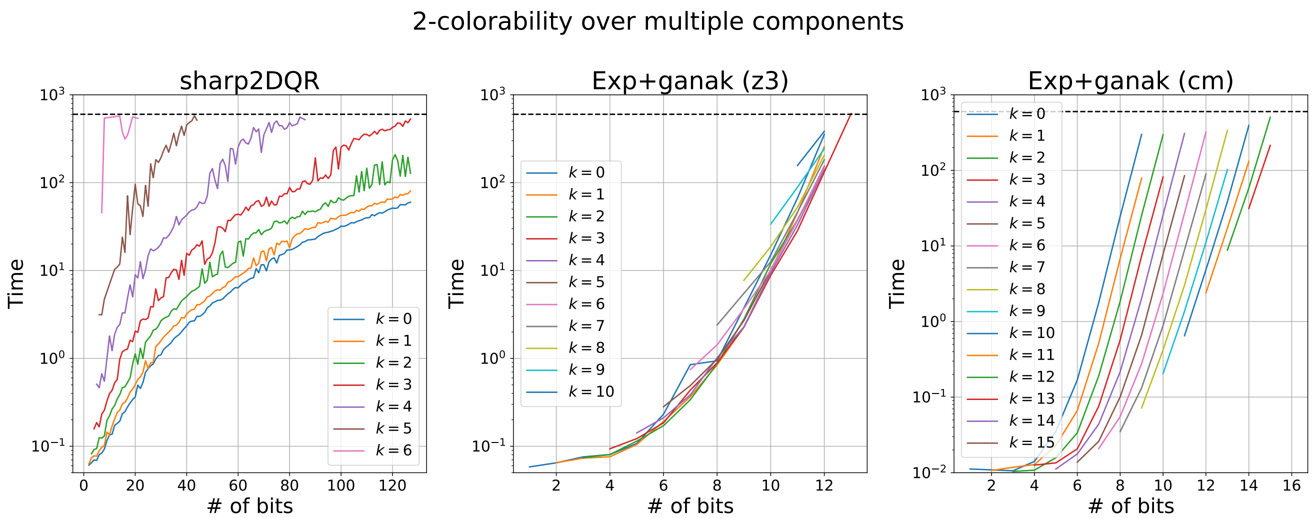

Figure 3: Performance of `sharp2DQR` and `Exp+ganak` on the counting the numbers of 2-colorings over $G_{n,k}$. The horizontal axis represents the number of bits in the graph, i.e. n.

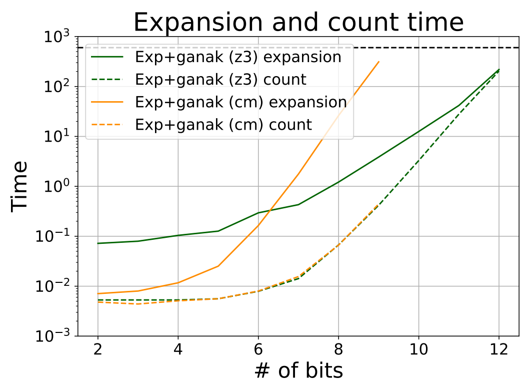

Figure 4: Time used on expansion and counting for `Exp+ganak` on 2-colorability instances.

Independent set

We consider the DQBF formula:

$$\text{IND-SET}_{n,k} \;:=\; \forall \bar{x} \forall \bar{x}' \exists y(\bar{x}) \exists y(\bar{x}').\; ((\bar{x} = \bar{x}') \to (y = y')) \wedge (C_{n,k} \to (\neg y \vee \neg y'))$$

The number of Skolem functions of $\text{IND-SET}_{n,k}$ is the number of (not necessarily maximal) independent set for $G_{n,k}$, which is $(2 \times 2^{2^{n-k-1}} - 1)^{2^k}$.

Figure 5 shows the experimental results. `sharp2DQR` can only handle instances with $n \leq 12$ and $n - k \leq 3$, while `Exp+ganak` with `z3` can handle all instances with $n \leq 12$, and with `cryptominisat`, `Exp+ganak` can handle instances up to $n = 16$. `sharp2DQR` performed worse in this set of instances because it has to do a lot of enumeration since every subset of an independent set is an independent set.

Encoding the 2-colorability and independent set instances with first order logic

We can encode both type of instances in first order logic with some labeling predicates. For example, for the graph $E_{n,k}$, consider the following sentence over the signature $\{U_1, \ldots, U_n, C\}$ where all of the predicates are unary.

$$\Psi_{n,k} := \forall x \exists y \left(\text{1-type}(y) \equiv \text{1-type}(x) + 1 \bmod 2^n\right) \;\wedge\; \forall x \forall y\; \tilde{E}_{n,k} \to (C(x) \neq C(y))$$

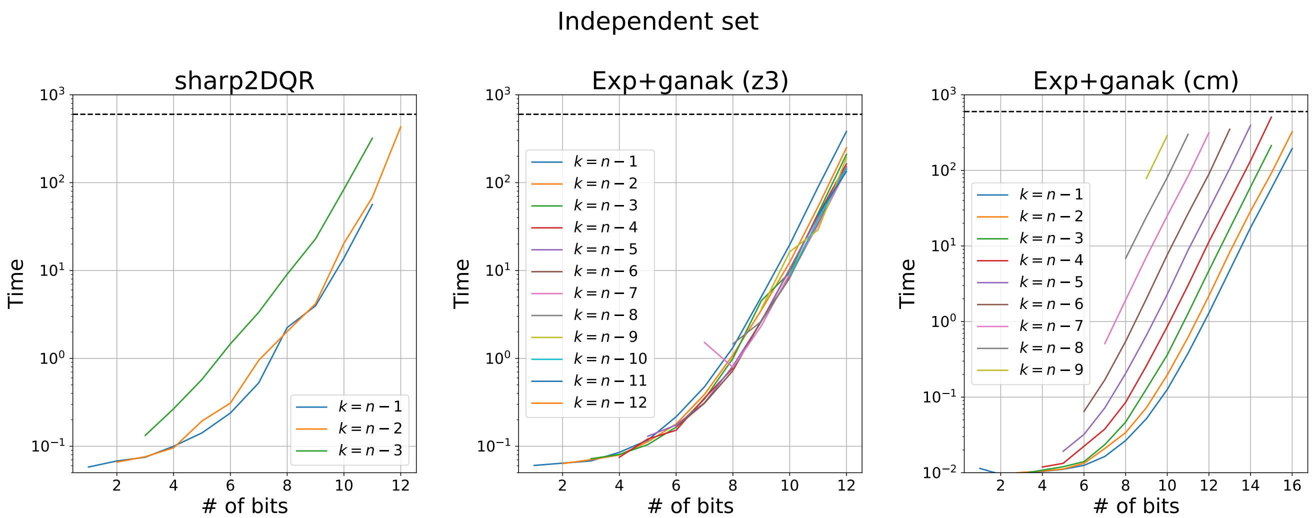

Figure 5: Performance of `sharp2DQR` and `Exp+ganak` on the counting the numbers of independent sets over $G_{n,k}$. The horizontal axis represents the number of bits in the graph, i.e. n.

where 1-type$(y) \equiv$ 1-type$(x) + 1 \bmod 2^n$ is the following formula:

$$\left(\bigwedge_{1\leq i\leq n} U_i(x) \wedge \neg U_i(y)\right) \vee \left(\bigvee_{1\leq i\leq n}\left(\neg U_i(x) \wedge U_i(y) \wedge \bigwedge_{1\leq j\leq i-1} U_j(x) \wedge \neg U_j(y) \wedge \bigwedge_{i+1\leq j\leq n} U_j(x) = U_j(y)\right)\right)$$

Intuitively, 1-type(x) is a maximal consistent subset of $\{U_1(x), \neg U_1(x), \ldots, U_n(x), \neg U_n(x)\}$ and 1-type(y) is a maximal consistent subset of $\{U_1(y), \neg U_1(y), \ldots, U_n(y), \neg U_n(y)\}$. We use 1-type to represent a number between 0 and $2^n - 1$, where $U_i(x)$ and $U_i(y)$ represent the i-th bit (of x and y). The atom $C(x)$ and $C(y)$ represent the color of the element x and y. The intention of the sentence $\forall x \exists y \left(\text{1-type}(y) \equiv \text{1-type}(x) + 1 \bmod 2^n\right)$ is to ensure all numbers between 0 and $2^n - 1$ exists.

The formula $\tilde{E}_{n,k}$ encodes the edge relation where we replace x_i with $U_i(x)$ and x'_i with $U_i(y)$ in $E_{n,k}$. The sentence $\forall x \forall y\ \tilde{E}_{n,k} \to (C(x) \neq C(y))$ states that no two adjacent elements have the same color.

Note that the number of models of $\Psi_{n,k}$ with size 2^n is the number of Skolem functions of TWO-COL$_{n,k}$ multipled by $(2^n)!$, due to the labeling of the elements in the models of $\Psi_{n,k}$. In our experiment, we tried counting the number of models of the resulting formula with `wfomc` (Wang 2025), where we set the domain size to 2^n. However, it can only solve instances with domain size up to 4.