#P is Sandwiched by One and Two #2DNF Calls: Is Subtraction Stronger Than We Thought?*

Max Bannach
Advanced Concepts Team
European Space Agency
Noordwijk, The Netherlands
max.bannach@esa.int

Erik D. Demaine
Computer Science and AI Lab
Massachusetts Institute of Technology
Cambridge, United States
demaine@mit.edu

Timothy Gomez
Computer Science and AI Lab
Massachusetts Institute of Technology
Cambridge, United States
tagomez7@mit.edu

Markus Hecher
Computer Science and AI Lab
Massachusetts Institute of Technology
Cambridge, United States
hecher@mit.edu

***Abstract*—The canonical class in the realm of counting complexity is $\#\mathrm{P}$. It is well known that the problem of counting the models of a propositional formula in disjunctive normal form (#DNF) is complete for $\#\mathrm{P}$ under Turing reductions. On the other hand, $\#\text{DNF} \in \mathrm{spanL}$ and $\mathrm{spanL} \subsetneq \#\mathrm{P}$ unless $\mathrm{NL} = \mathrm{NP}$. Hence, the class of functions logspace-reducible to #DNF is a strict subset of $\#\mathrm{P}$ under plausible complexity-theoretic assumptions. By contrast, we show that *two* calls to a (restricted) #2DNF oracle suffice to capture gapP, namely, that the logspace many-one closure of the subtraction between the results of two #2DNF calls *is* gapP. Because $\#\mathrm{P} \subsetneq \mathrm{gapP}$, $\#\mathrm{P}$ is strictly contained between one and two #2DNF oracle calls.**

Surprisingly, the propositional formulas needed in both calls are *linear-time* computable, and the reduction preserves interesting structural as well as symmetry properties, leading to algorithmic applications. We show that a *single subtraction* suffices to compensate for the absence of negation while still capturing gapP, i.e., our results carry over to the monotone fragments of #2SAT and #2DNF. Given our fine-grained linear-time reduction, as a consequence of our result, we obtain a *sparsification lemma* for #2SAT, which has only been known for kSAT with $k \geq 3$ and corresponding counting versions.

We further show that both #2DNF calls can be combined into a single call if we allow a little postprocessing (computable by AC^0- or TC^0-circuits). Consequently, we derive refined versions of Toda's Theorem: $\mathrm{PH} \subseteq [\#\text{MON2SAT}]^{\log}_{\mathrm{TC}^0} = [\#\text{MON2DNF}]^{\log}_{\mathrm{TC}^0}$ and $\mathrm{PH} \subseteq [\#\text{IMPL2SAT}]^{\log}_{\mathrm{AC}^0}$. Our route to these results is via structure-preserving reductions that preserve parameters like treewidth up to an additive overhead. The absence of multiplicative overhead indeed immediately yields parameterized SETH-tight lower bounds for counting.

***Index Terms*—counting complexity, sharp-p, span-l, satisfiability, sharp-sat, SETH, fixed-parameter tractability, treewidth, linear-time reduction, lower bound**

I. Introduction

The function problem #SAT asks, given a propositional formula φ in conjunctive normal form (a CNF), how many of the 2^n possible assignments[1] $\beta \subseteq \mathrm{vars}(\varphi)$ satisfy φ, that is, the task is to determine the number $\#(\varphi)$ of *models* of φ. It is well known that #SAT is complete for $\#\mathrm{P}$, the class of functions definable as the number of accepting paths of a polynomial-time nondeterministic Turing machine. In fact, #SAT is $\#\mathrm{P}$-complete under logspace many-one reduction, because the Cook-Levin construction is solution preserving [41, Lemma 3.2]. Denoting the closure under logspace many-one reduction[2] by $[\cdot]^{\log}$, we *characterize* $\#\mathrm{P}$ as:

Observation 1. $\#\mathrm{P} = [\#\text{SAT}]^{\log} = [\#3\text{SAT}]^{\log}$.

In stark contrast to the decision version of the problem, #SAT remains hard even for heavily restricted fragments of propositional logic. For instance, #SAT trivially reduces to #DNF, which asks for the number of models of a formula in *disjunctive* normal form: observe that $\neg\varphi$ is a DNF and that $\#(\varphi) = 2^n - \#(\neg\varphi)$. This reduction has two additional features: *(i)* we require only *one call* to a #DNF oracle, and *(ii)* we need to perform *one subtraction* in a postprocessing step after querying the oracle. That is, we did *not* show a logspace many-one reduction from #SAT to #DNF, but from #SAT to the problem of computing $2^n - \#(\psi)$ for a formula $\psi \in$ #DNF, which we denote as $2^n -$ #DNF:

Observation 2. $\#\mathrm{P} = [2^n - \#\text{DNF}]^{\log}$.

One may be tempted to think that this is just a slight technicality, but in fact this *subtraction is crucial:* #DNF lies in spanL, the class functions expressible as the *span of a nondeterministic logspace* Turing machine, i.e., the number of distinct outputs that an NL-transducer can produce [2]. We know $\mathrm{spanL} \subsetneq \#\mathrm{P}$ unless $\mathrm{NL} = \mathrm{NP}$ [2,

* Author names are given in alphabetical order.

[1] We represent an *assignment* by $\beta \subseteq \mathrm{vars}(\varphi)$, interpreted as the subset of *variables* assigned to true.

[2] Many-one reductions (no postprocessing) imply parsimony. A reduction is *c-monious* if it preserves the solution count up to a factor c.

Proposition 4.10] and, hence, $\#\mathrm{P} = [\#\mathrm{DNF}]^{\log}$ is unlikely under plausible complexity-theoretic assumptions.

The quest for understanding the complexity of #2SAT and #2DNF. The examples illustrate that counting remains hard on syntactically restricted formulas, and that they do so by surprisingly simple reductions. One usual suspect that seems to be missing is #2SAT, for which one would expect a similar reduction. The seminal work by Valiant [41], [42] proved that #2SAT is #P-hard by a sophisticated chain of reductions from #SAT, via several variations of the problem of computing a permanent, to the task of counting matchings in graphs, and then finally to #2SAT. This chain of reductions results in a time effort of at least $\Omega(n^3 \log n)$ because of formulas of size $\Omega(n^2)$ (the reduction from computing perfect matchings to imperfect matchings [42, Step 6 in Theorem 1]) and a *polynomial number* of oracle calls as well as the ability to postprocess the results *modulo* a polynomially bounded number [41, Proposition 3.4]. The insight that even a simple "minus" in postprocessing can have dramatic impacts on the complexity of counting problems raises the question of how much of the complexity of #2SAT is "hidden" by this seemingly involved reduction. This question leads to the quest for a direct reduction from #SAT to #2SAT and, in the light of Observation 2, to #2DNF. *What* makes #2SAT and #2DNF hard?

There are multiple complexity classes to characterize these so-called "easy to decide and hard to count problems" [6]. Almost all of these classes collapse under Turing reductions, making it necessary to study parsimony. We mention two classes here: totP and spanL. The first is the class of counting problems corresponding to the number of *all* path of a polynomial time NDTM. It was proven in [33] that totP is exactly the set of problems which (1) have an easy decision version[3] and (2) are self-reducible. Two subclasses of totP were studied in [8] with second order logic showing connections to generalizations of #2SAT. A number of totP-complete problems were found in [3]. The class spanL is contained in totP. It is known that spanL admits a *fully polynomial randomized approximation scheme* (FPRAS) via counting the number of strings of length n accepted by an NFA [2], [4], [30]. Since this reduction is parsimonious it preserves the approximation. On the other hand, totP contains many important problems such as #2SAT and #PERFECT-MATCHING, and the former is sufficient to show that totP is *not* a subset of FPRAS. Neither #2SAT nor #DNF are known to be complete for either class.

[3]This class is called #PE. The subset of #P with easy decision versions.

A. Contribution I: Reducing #SAT to Two Calls of #2DNF

We provide a new reduction from #SAT to *two* calls to a #2SAT or #2DNF oracle. Crucially, we only need a *single subtraction* to combine the results (no involved postprocessing or modulo computations). All our reductions are logspace computable and, thus, we can phrase our #2SAT reduction as follows:

Theorem 3. $\#\mathrm{SAT} \in [\#2\mathrm{SAT} - \#2\mathrm{SAT}]^{\log} = [\#2\mathrm{DNF} - \#2\mathrm{DNF}]^{\log}$.

Note that, since #2SAT does not admit an FPRAS under common assumptions [37], we have $\#2\mathrm{SAT} \notin [\#2\mathrm{DNF}]^{\log}$ and, hence, an Immerman-Szelepcsényi-type theorem does *not hold* in the counting world. Theorem 3, however, establishes a theorem of this type if *two* calls are permitted. In fact, we prove a stronger form of the first part of the theorem: we reduce to restricted versions of #2SAT; the reduction can be implemented either in logspace or in linear time; and it preserves important structural and symmetry properties of the input.

Lemma 4 (Main Lemma)**.** *There is a { linear-time, logspace } algorithm mapping a* CNF φ *and a corresponding tree decomposition to* CNF*s* ψ_1, ψ_2 *with at most two variables per clause such that, for* $\rho \in \{\mathrm{tw}, \mathrm{itw}, \mathrm{bw}, \mathrm{ibw}\}$,[4]

$$\#(\varphi) = \#(\psi_1) - \#(\psi_2) \quad \textit{and}$$
$$\max\{\rho(\psi_1), \rho(\psi_2)\} \leq \alpha \cdot \rho(\varphi) + 14 \textit{ with } \alpha = 1.$$

For $\rho \in \{\mathrm{tw}, \mathrm{itw}\}$*, the resulting formulas can be restricted to the following fragments:*

(A) $\alpha = 1$ *and* ψ_1, ψ_2 *are* monotone*, i.e., do not contain negations; or*

(B) $\alpha = 3$ *and* ψ_1, ψ_2 *comprise binary* implications *and are* cubic *and* bipartite*, i.e., every variable occurs at most three times and the primal graph does not contain an odd cycle.*

The second part of Theorem 3 follows because $[\#2\mathrm{SAT} - \#2\mathrm{SAT}]^{\log}$ and $[\#2\mathrm{DNF} - \#2\mathrm{DNF}]^{\log}$ both turn out to be precisely gapP. We discuss this further in the next subsection. While Lemma 4 is the key to all our contributions, a direct consequence is the following:

Corollary 5. $\#\mathrm{P} \subsetneq [\#\mathrm{MON2SAT} - \#\mathrm{MON2SAT}]^{\log} \subseteq \mathrm{gapP}$.

Proof. Containment follows from Lemma 4; it must be strict as functions in #P cannot map to negative numbers but functions in $[\#\mathrm{MON2SAT} - \#\mathrm{MON2SAT}]^{\log}$ can. ☐

Interestingly, we do not expect to improve the reduction of Lemma 4 for planarity while keeping linear time.

[4]Here $\mathrm{tw}(\varphi), \mathrm{itw}(\varphi), \mathrm{bw}(\varphi), \mathrm{ibw}(\varphi)$ denote treewidth and bandwidth of two different graphs associated with formula φ; see Section I-D.

Proposition 6. *Unless* ETH *fails, we can not have a c-monious*[2] *linear-time Turing reduction from* 3SAT *to* #PLANAR3SAT.

Proof. Suppose towards a contradiction that such a reduction exists. Then, we can decide 3SAT with n variables via linear many #PLANAR3SAT calls. However, each of these calls can be decided in time $2^{\mathcal{O}(\sqrt{n})} \cdot n^{\mathcal{O}(1)}$, as planar graphs can be grid-embedded; in the worst case both dimensions are roughly equal ($\mathcal{O}(\sqrt{n}) \times \mathcal{O}(\sqrt{n})$) since in a grid the treewidth is the smaller of both. This contradicts ETH, deciding 3SAT in $2^{o(n)} \cdot n^{\mathcal{O}(1)}$. □

B. Contribution II: New Characterization of GapP

Because subtracting the model counts of two monotone formulas is enough to capture $\#\mathrm{P}$, the natural next question is what is needed to capture gapP. We show that even this class can be characterized by two calls to oracles of restricted fragments of #2SAT or #2DNF. In the following theorem, $\#\mathrm{P}-\#\mathrm{P}$ (respectively $\mathrm{spanL}-\mathrm{spanL}$) refers to the subtraction of the results of two $\#\mathrm{P}$ (respectively spanL) oracle calls. Since it is open whether #DNF is spanL-hard, and since it is *not expected* [37, Theorem 2] for #2SAT to be in spanL (unless $\mathrm{RP} = \mathrm{NP}$), this makes our characterization of $\#\mathrm{P} - \#\mathrm{P}$ via $[\#2\mathrm{SAT} - \#2\mathrm{SAT}]^{\log}$ *and its dual* $[\#2\mathrm{DNF} - \#2\mathrm{DNF}]^{\log}$ significant; the result holds even in the absence of negation. Below, #IMPL2SAT is strictly in #HORN2SAT and #0,1-2DNF is its dual over DNF.

Theorem 7 (Characterization of GapP)**.**

$$\begin{aligned}&\mathrm{gapP}\\ &= [\#2\mathrm{SAT}-\#2\mathrm{SAT}]^{\log} = [\#\mathrm{IMPL2SAT}-\#\mathrm{IMPL2SAT}]^{\log}\\ &= [\#0,1\text{-}2\mathrm{DNF}-\#0,1\text{-}2\mathrm{DNF}]^{\log} = [\#2\mathrm{DNF}-\#2\mathrm{DNF}]^{\log}\\ &= [\#\mathrm{DNF}-\#\mathrm{DNF}]^{\log} = [\#\mathrm{MON2SAT}-\#\mathrm{MON2SAT}]^{\log}\\ &= [\#\mathrm{MON2DNF}-\#\mathrm{MON2DNF}]^{\log} = \mathrm{spanL}-\mathrm{spanL}.\end{aligned}$$

The characterization extends to cubic *and* bipartite *restrictions of* #IMPL2SAT *and* #0,1-2DNF*; even if both formulas use the same variables and differ by only one literal/variable occurrence.*

This result illustrates the *power of subtraction*, which by the theorem compensates for both the absence of negation *and* clauses of size at least three. Since it is known that $\#2\mathrm{DNF} \in \mathrm{spanL}$ with $\mathrm{spanL} \subsetneq \#\mathrm{P}$ (unless $\mathrm{NL} = \mathrm{NP}$ [2, Proposition 4.10]) and $\#\mathrm{P} \subsetneq \mathrm{gapP}$ by Observation 5, Theorem 7 implies that $\#\mathrm{P}$ is *strictly sandwiched* between one and two #2DNF oracle calls:

Corollary 8. $[\#2\mathrm{DNF}]^{\log} \subsetneq \#\mathrm{P} \subsetneq \mathrm{gapP} = [\#2\mathrm{DNF} - \#2\mathrm{DNF}]^{\log}$ *unless* $\mathrm{NL} = \mathrm{NP}$.

Figure 1 depicts an overview of counting classes and their logical description under logspace many-one reductions. We added the *positive* part of gapP defined as $\mathrm{gapP}^+ = \{ f \in \mathrm{gapP} \mid \forall x : f(x) \geq 0 \}$, which, unless $\mathrm{UP} = \mathrm{SPP}$, contains functions that are not in $\#\mathrm{P}$ [32].

Observation 9. $\#\mathrm{P} \subseteq \mathrm{gapP}^+ \subsetneq \mathrm{gapP}$ *and* $\#\mathrm{P} \subsetneq \mathrm{gapP}^+$ *unless* $\mathrm{UP} = \mathrm{SPP}$.

Figure 1 also incorporates the following lemma, which observes that logspace-computable c-monious reductions between #2SAT and #3SAT are not possible unless NL and NP collapse.

Lemma 10. $[\#2\mathrm{SAT}]^{\log} \subsetneq [\#3\mathrm{SAT}]^{\log}$ *unless* $\mathrm{NL} = \mathrm{NP}$.

Proof. We show a stronger result. Suppose we have a *c-monious* logspace reduction R from #3SAT to #2SAT for a positive integer constant c, i.e., R changes the number of solutions by precisely a multiplicative factor of c. Then the following algorithm decides 3SAT in NL: On input φ, first compute $R(\varphi)$ with $\#(\varphi) = c \cdot \#(R(\varphi))$, which is possible by assumption and since logspace is closed under composition. If φ is unsatisfiable, we have $\#(\varphi) = c \cdot \#(R(\varphi)) = 0$, and otherwise we have $\#(R(\varphi)) > 0$ since $c > 0$. We decide whether $\#(R(\varphi)) > 0$ by solving 2SAT, which is possible since NL is closed under complement [27]. □

C. Contribution III: New Characterization of the Polynomial Hierarchy

Toda's celebrated theorem [39] states that the whole polynomial hierarchy can be solved by a polynomial-time Turing machine using a single call to a $\#\mathrm{P}$ oracle. In fact, a spanL oracle suffices [2, Corollary 4.11]:

Observation 11. $\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}[1]} \subseteq \mathrm{P}^{\#\mathrm{P}} = \mathrm{P}^{\mathrm{spanL}}$.

In the framework of logspace many-one reductions, the emerging question is how much computation needs to be carried out by the polynomial-time Turing machine. Is it sufficient to "just" prepare the oracle call, or is significant postprocessing necessary? Below we prove that we can combine the two calls to #2SAT oracles needed in Theorem 3 into a *single call* if we allow *divisions* afterward. More precisely, we show that $\#\mathrm{P}$ can be simulated by a logspace reduction to #MON2SAT followed by TC^0 postprocessing:

Theorem 12. $\mathrm{gapP} \subseteq [\#\mathrm{MON2SAT}]^{\log}_{\mathrm{TC}^0} = [\#\mathrm{MON2DNF}]^{\log}_{\mathrm{TC}^0}$.

A crucial part of the postprocessing done by the TC^0 circuit in Theorem 12 is *division* and, thus, we do not expect to be able to lower the postprocessing power since division is TC^0-complete [25] and $\mathrm{AC}^0 \subsetneq \mathrm{TC}^0$ [23]. However, if we allow a slightly more powerful fragment of propositional logic, we can prepare a count that we just divide by a power of 2, which *is* possible in AC^0:

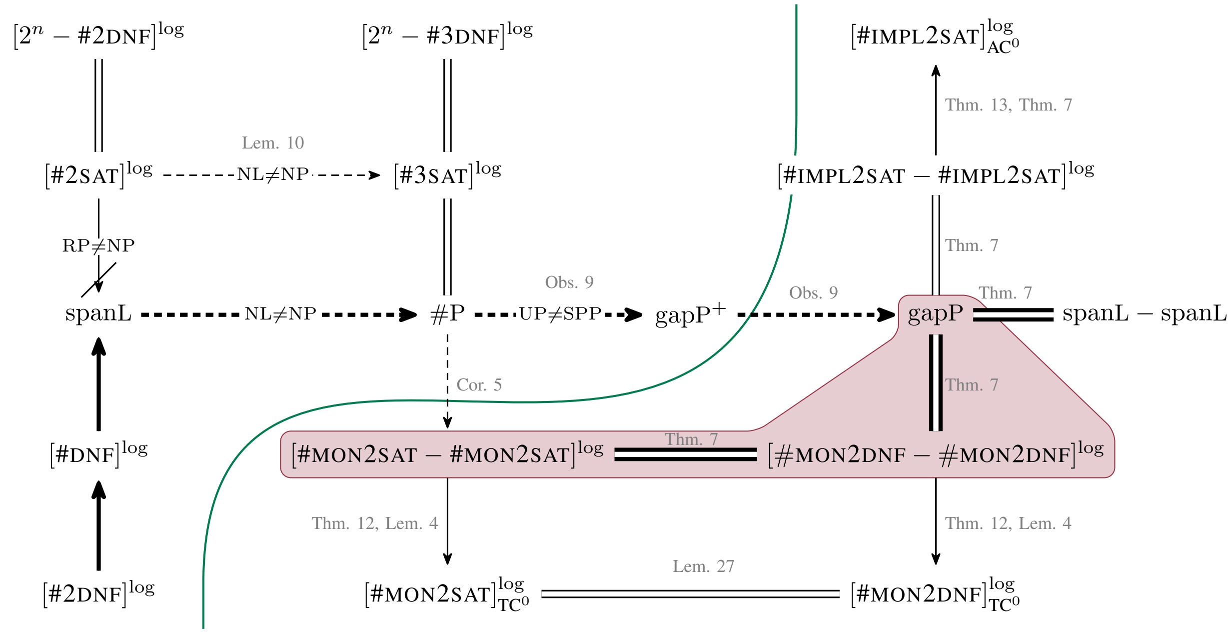


Fig. 1. Overview of complexity classes considered in this paper. An arrow $A \longrightarrow B$ indicates $A \subseteq B$, a dashed arrow means that the implication is strict under the assumption shown on the line (e.g., $\mathrm{NL} \neq \mathrm{NP}$ implies $\mathrm{spanL} \subsetneq \#\mathrm{P}$), and $A = B$ stands for $A = B$. If the arrow tip is crossed out, A is not contained in B. We use $[\cdot]^{\log}$ to indicate the logspace closure of the problem in brackets (e.g., $[\#3\mathrm{SAT}]^{\log}$ is the logspace (many-one, parsimonious) closure of #3SAT). The shorthand A − B indicates that two oracle calls are allowed (with potentially different instances), and the result is the difference between both calls. A class in the subscript such as in $[\#\mathrm{Mon2SAT}]^{\log}_{\mathrm{TC}^0}$ indicates that TC^0 postprocessing is allowed *after* the oracle call. Sets to the left of the green line contain only positive functions, while sets on the right side contain functions that map to negative numbers. Hence, the left side is strictly contained in the right. **Thick edges** are fundamental, highlighting differences between one and two #2DNF calls. The red area marks central insights of our contribution.

Theorem 13. $\mathrm{gapP} \subseteq [\#\mathrm{IMPL2SAT}]^{\log}_{\mathrm{AC}^0} = [\#0,1\text{-}2\mathrm{DNF}]^{\log}_{\mathrm{AC}^0}$. *This statement holds even if* #IMPL2SAT *is restricted to cubic and bipartite formulas.*

Finally, we can use these two results to obtain a stronger variant of Toda's celebrated result [39] using logspace reductions to counting problems of restricted fragments of propositional logic (even contained in #2DNF) with only little postprocessing:

Theorem 14 (Characterization of PH)**.**

$$\begin{array}{ccccc} & [\#\mathrm{MON2SAT}]^{\log}_{\mathrm{TC}^0} & = & [\#\mathrm{MON2DNF}]^{\log}_{\mathrm{TC}^0} & \\ \rotatebox{45}{$\subseteq$} & & & & \rotatebox{-45}{$\subseteq$} \\ \mathrm{PH} & & & & \mathrm{P}^{\#2\mathrm{DNF}[1]} \\ \rotatebox{-45}{$\subseteq$} & & & & \rotatebox{45}{$\subseteq$} \\ & [\#\mathrm{IMPL2SAT}]^{\log}_{\mathrm{AC}^0} & = & [\#0,1\text{-}2\mathrm{DNF}]^{\log}_{\mathrm{AC}^0} & \end{array}$$

D. Contribution IV: New Upper and Lower Bounds for #SAT

Finally, we observe that the reductions used to prove Theorem 3 can be implemented in linear time and that they preserve important structural parameters of the input such as its treewidth (the details of Lemma 4). The lemma has some immediate algorithmic consequences. Interestingly, we obtain tight (SETH-based) lower bounds, *via parameterized complexity*, as we establish strong parameterized guarantees with *only additive overhead* (already a multiplicative factor larger than 1 is problematic). Indeed, without this parameterized route, obtaining such tight bounds is challenging, as there exists an $\mathcal{O}(1.3^n)$ algorithm [16].

First note that fine distinctions can be made when defining structural properties of propositional formulas. Usually, parameters such as the treewidth $\mathrm{tw}(\varphi)$ are defined over the *primal graph,* which is the graph that contains a vertex for every variable and that connects two variables if they appear together in a clause. Another graphical representation of a formula is the *incidence graph,* which contains a vertex for every variable *and every clause* and that connects two vertices if the variable appears in the clause. The latter representation gives rise to the definition of *incidence treewidth* $\mathrm{itw}(\varphi)$ for which it is known that $\mathrm{itw}(\varphi) \leq \mathrm{tw}(\varphi) + 1$ [11, Chapter 17].

It is relatively easy to show that $\#(\varphi)$ can be computed with $O\big(2^{\mathrm{tw}(\varphi)}|\varphi|\big)$ or $O\big(4^{\mathrm{itw}(\varphi)}|\varphi|\big)$ arithmetic operations. It was a long-standing open problem whether the exponential dependency on $\mathrm{itw}(\varphi)$ can be improved to $O\big(2^{\mathrm{tw}(\varphi)}|\varphi|\big)$, which Slivovsky and Szeider [36] answered affirmatively with an involved algorithm utilizing zeta and Möbius transforms to compute covering products. We obtain the result as a corollary from Lemma 4 because our reduction to #2SAT increases the incidence treewidth *only by an additive constant*:

Corollary 15. *There is an algorithm computing* $\#(\varphi)$ *in* $O(2^{\mathrm{itw}(\varphi)}|\varphi|)$ *arithmetic operations.*

Proof. By Lemma 4, we can reduce a CNF φ to 2CNFs ψ_1 and ψ_2 with $\text{itw}(\psi_i) \leq \text{itw}(\varphi) + 14$. In the incidence graph of a 2CNF, all vertices corresponding to clauses have maximum degree 2. By the *almost simplicial rule*, contracting such a vertex to one of its neighbors cannot increase the treewidth past 2 [14]. However, contracting all vertices corresponding to a clause to one of their neighbors yields exactly the primal graph, hence, we have $\text{tw}(\psi_i) \leq \text{itw}(\psi_i) + 1 \leq \text{itw}(\varphi) + 15$. Finally, we compute $\#(\psi_i)$ by dynamic programming over a tree decomposition of the primal graph, requiring $O\big(2^{\text{tw}(\psi_i)}|\psi_i|\big)$ arithmetic operations [12], [35]. $\square$

SETH-Tight Lower Bounds.: On the other hand, since the reduction preserves structural properties up to an *additive* constant factor, we can complement the upper bound with a tight lower bound under the (strong) exponential time hypothesis (SETH) [28].

Theorem 16 (SETH LB). *Under* SETH, $\#(\varphi)$ *cannot be computed with* $o\big(2^{\rho}\big) \cdot |\varphi|^{O(1)}$ *arithmetic operations on formulas* φ *with at most two variables per clause and* $\rho \in \{\text{bw}(\varphi), \text{ibw}(\varphi), \text{tw}(\varphi), \text{itw}(\varphi)\}$. *Results extend to (A) bipartite monotone formulas for* $\rho \in \{\text{tw}(\varphi), \text{itw}(\varphi)\}$.

Proof. SETH [28] implies that we cannot decide s-SAT in time $o(2^{\rho}) \cdot |\varphi|^{O(1)}$. (A) follows from strong parameter guarantees of Lemma 4 and a slightly modified reduction. Thereby, in Equations (14), we replace clauses of the form $\top_v \vee \bot_v$ (preventing bipartiteness) by $\top_v \vee a_1$, $\bot_v \vee b_1$, …, $\top_v \vee a_c$, $\bot_v \vee b_c$ for fresh variables a_i, b_i, constant $c{>}s$. This enables integer-dividing the resulting count by $2^{c|\text{vars}(\varphi)|}$, to recover number of solutions. $\square$

Note that for strong SETH-based bounds, already a linear factor as in Lemma 4(B) is problematic. However, under ETH we obtain these constant-degree results.

Theorem 17 (ETH LB). *Unless* ETH *fails,* $\#(\varphi)$ *cannot be computed with* $\big(2^{o(\rho)}\big) \cdot |\varphi|^{O(1)}$ *arithmetic operations on formulas* φ *with at most two variables per clause, where* $\rho \in \{\text{bw}(\varphi), \text{ibw}(\varphi), \text{tw}(\varphi), \text{itw}(\varphi)\}$. *The result still holds for* $\rho \in \{\text{tw}(\varphi), \text{itw}(\varphi)\}$, *and (A) bipartite constant-degree formulas without negation or (B) bipartite implication formulas of degree* 3.

Proof. For (A) and (B), we normalize to at most 3 occurrences (degree) per variable, resulting in a linear increase of $\text{tw}(\varphi)$. For (A), the proof of Theorem 16 causes indeed only a constant-degree blowup, (B) holds by Lemma 4. By Cor. 15, the result holds for $\text{itw}(\varphi)$. $\square$

In addition, we obtain the following strong non-parameterized bound, which significantly improves a lower bound of the literature [15, Corollary 4.4], as *our reduction preserves parameters*. Without our parameterized detour, we would *not directly obtain* such a strong bound below. We derive stronger results than [15, Corollary 4.4] by sparsifying first [28].

Corollary 18. *Under* ETH, $\#(\varphi)$ *cannot be computed in time* $\big(2^{o(n)}\big) \cdot |\varphi|^{O(1)}$ *on formulas* φ *with at most two variables per clause,* n *variables, and* $\mathcal{O}(n)$ *clauses. The result holds on (A) bipartite constant-degree formulas without negation and (B) bipartite degree-*3 *implications.*

All bounds carry over to stronger results under weaker assumptions than ETH and SETH, namely counting-based versions #ETH [17] and #SETH [21], respectively.

E. Related Work

Recently, Laakkonen, Meichanetzidis, and Wetering [29] also provided a new reduction from #SAT to #2SAT using the ZH-calculus. Their work focuses on producing a simple reduction that is representable in a pictorial way. In comparison, Laakkonen, Meichanetzidis, and Wetering reduce to a *single* #2SAT-instance, but require *modulo computations* in postprocessing; while we are parsimonious, requiring a *single subtraction*.

Another downside of the ZH-based reduction is a *quadratic* blow-up: If the original formula φ has n variables and m clauses, the produced formula will have size $O(n + mn)$. This blow-up quickly accumulates if we perform further reductions. For instance, the same set of authors also provided a reduction from #2SAT to #MON2SAT that maps an instance with n variables and m clauses to a formula with $O(n + mn^2)$ variables and $O(m + mn^2)$ clauses (Lemma 5 in [29]). Hence, the whole reduction from #SAT to #MON2SAT constructs $O(n^3m^3)$ clauses. In contrast, our reductions produce instances of *linear* size, i.e., $O(n + m)$, for all the restricted fragments mentioned.

There is an extensive study on closure properties of #P and other counting complexity classes, see, e.g., [22], [26], [31], [38]. There are also interesting findings regarding the closure of PP under intersection [10], which uses closure properties of #P. Closure properties of $\#\text{AC}^0$ have been studied in [1] ($\#\text{AC}^0$ is the set of functions computable by arithmetic circuits). They show negative results for functions such as max and division by 3. Connections are shown between $\#\text{AC}^0$, gapAC^0, and threshold circuits as well. The class #P is also closed under so-called *subtractive reductions* [18] along with other variants of counting classes in the polynomial hierarchy. These reductions use a different form of subtraction, as they are based on set difference, but not on the more general subtraction of counts (numbers).

There have been many other classes [9] defined similarly to our postprocessing strategy, such as $mod_k\mathrm{P}$ [24], i.e., counting the accepting paths of an NP machine and outputing whether the result is divisible by k. For $k = 2$, this problem is known as $\oplus\mathrm{P}$ [34], which contains for instance graph isomorphism [5].

F. Structure of the Paper

In Section II we provide an overview of our techniques, which is followed by concluding remarks and discussions in Section III. Section IV recalls preliminaries and defines common notation. Then, Section V focuses on Contribution I and establishes our main reduction from #SAT to (two calls of) #2SAT, as well as Theorem 3. In Section V-A, we show claimed structural properties leading to Contribution IV, followed by extensions of our reduction to restricted variants in Sections V-B. Then, Section VI covers Contribution II, thereby showing consequences of our reductions (also for #2DNF) and its relationship to gapP. Finally, in Section VII we show Contribution III, where we demonstrate how to reduce #SAT to a single #2SAT (or #2DNF) oracle call, followed by AC^0 or TC^0 postprocessing.

II. Overview of Used Techniques

The backbone of our reduction is the fact that #SAT can be reduced to *weighted* #2SAT by encoding the *inclusion-exclusion principle.* In *weighted* #SAT, also known as #WSAT or WMC, the input is a weighted CNF (a WCNF), i.e., a CNF ψ together with weights $w\colon \mathrm{vars}(\psi) \to \mathbb{R}$. The goal is to compute the *weighted* (or *scaled*) count:
$$\#_w(\psi) := \sum_{\substack{\beta \subseteq \mathrm{vars}(\psi)\\ \beta \models \psi}} \prod_{x \in \beta} w(x).$$

The reduction sets $w(x) = 1$ for all $x \in \mathrm{vars}(\varphi)$ and introduces for every clause $c = \ell_1 \vee \cdots \vee \ell_k$ a fresh variable $\underset{\cdot}{c}$ with $w(\underset{\cdot}{c}) = -1$ as well as the new set of clauses $\bigwedge_{i=1}^{k}(\neg\underset{\cdot}{c} \vee \neg\ell_i)$. Intuitively, the variable $\underset{\cdot}{c}$ indicates that the clause c is *not* satisfied by the assignment, i.e., if we set $\underset{\cdot}{c}$ to true we have to falsify all literals in c. Let the resulting WCNF be ψ, then there are 2^n assignments for ψ that contribute 1 to the weighted count (those setting $\underset{\cdot}{c}$ variables to false).

On the other hand, every assignment that sets exactly one $\underset{\cdot}{c}$ to true (i.e., that falsifies at least one clause in φ) will contribute -1 to the weighted count (and, crucially is *not* a model of φ). Hence, from 2^n we will automatically *subtract* all assignments that do not satisfy *one* clause. All assignments that falsify *two* clauses will again contribute one (because the -1 cancel out in the product), all assignments that falsify three clauses will subtract one, for four clauses they add one, and so one. By the inclusion-exclusion principle we conclude: $\#(\varphi) = \#_w(\psi) = 2^n - |\{\, \beta \mid \beta \subseteq \mathrm{vars}(\varphi) \text{ with } \beta \not\models \varphi \,\}|$.

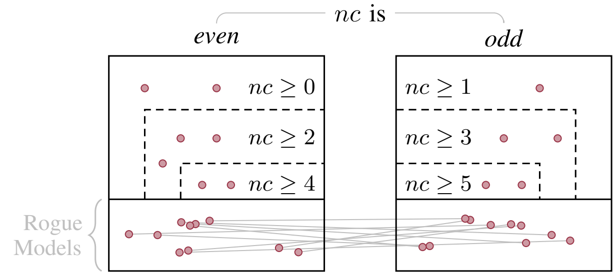


Fig. 2. Simplified illustration of the relation of satisfying assignments (models) between our constructed formulas. Each box indicates a set of models of a formula. Models are represented by circles. The models can be separated into non-rogue models on the top and rogue models on the bottom. Rogue models have a line indicating they have a bijection with a model in the other formula. Non-rogue models are divided based on dissatisfying at least nc many clauses. Dotted lines indicate the set of models inside the line is a subset of the larger set.

Example 19. *Consider $\varphi = c_1 \wedge c_2 \wedge c_3$ with $c_1 = \neg a \vee b \vee c$, $c_2 = a \vee \neg b \vee c$, and $c_3 = \neg c$. The inclusion–exclusion reduction produces the formula:*
$$\psi = (\neg\underset{\cdot}{c}_1 \vee a) \wedge (\neg\underset{\cdot}{c}_1 \vee \neg b) \wedge (\neg\underset{\cdot}{c}_1 \vee \neg c) \wedge (\neg\underset{\cdot}{c}_2 \vee \neg a) \wedge (\neg\underset{\cdot}{c}_2 \vee b) \wedge (\neg\underset{\cdot}{c}_2 \vee \neg c) \wedge (\neg\underset{\cdot}{c}_3 \vee c).$$
with weight $w(\underset{\cdot}{c}_1) = w(\underset{\cdot}{c}_2) = w(\underset{\cdot}{c}_3) = -1$ and $w(a) = w(b) = w(c) = 1$. The number of assignments that falsify c_1 is 1, for c_2 is 1, and 4 for c_3. There are no assignments that falsify c_1 and c_2 or c_1 and c_3 simultaneously, respectively; there is also no assignment that falsifies c_2 and c_3. Since, finally, there is no assignment that falsifies all three clauses, we obtain $\#_w(\psi) = 2^3 - 1 - 1 - 4 + 0 + 0 + 0 - 0 = 2 = \#(\varphi)$.

A fault-tolerant version of inclusion-exclusion. While inclusion-exclusion provides a reduction from #SAT to weighted #2SAT, we have the new problem of getting away with *negative* weights. We indirectly realize inclusion-exclusion (and, thus, shave off weights) with a novel *fault-tolerant version of inclusion-exclusion*. The idea is that the first count may make errors (e.g., over- or under-count), but these errors can be carefully controlled to be well-behaved. We then count these errors using the second formula such that subtracting the results of both calls results in the correct model count. We call these errors *rogue models* and outline the concept in Figure 2. While we cannot properly quantify (and express) these rogue models via one counting operation, aligning rogue models in a symmetric way allows us to “redo” errors, which then indirectly paves the way for separating models from rogue models. By construction, the formulas used in both calls are *almost identical* (just a single fact differs). In fact, both formulas share the *same* number of variables, which immediately gives *closure under negation*: $\#(\psi_1) - \#(\psi_2) = 2^n - \#(\neg\psi_1) - (2^n - \#(\neg\psi_2)) = \#(\neg\psi_2) - \#(\neg\psi_1)$. This yields further results and insights even for fragments in which "padding" might not be expressible. The more restrictive the #2CNF (#2DNF) fragment gets, the easier it is to break this symmetry, potentially yielding incorrect results. We guide the two calls along a structural representation of the formula (say, a tree decomposition), but do not directly utilize the width of the decomposition (e.g., the reductions work for unstructured instances as well). See Figure 3 for an illustration, which highlights functional dependencies.

How to simulate PH with a single #Mon2DNF call? Theorems 12 and 13 can be proven using the idea of creating a new formula φ by merging all the clauses of two formulas φ_1 and φ_2 with new variables. The key technique used is a way to *switch* between the two formulas resulting in $\#(\varphi) = f(\#(\varphi_1), \#(\varphi_2))$. It is easy to create a reduction that results in $\#(\varphi_1) \cdot \#(\varphi_2)$; however, due to the commutative property of multiplication, we cannot tell which count is from which formula. Thus, we design specific *switch* constructions that overcome this limitation by creating default assignments that fix the variables of one formula while allowing the others to be set freely. For restricted fragments, this is indeed challenging. For #IMPL2SAT to capture gapP we can encode both counts in the function $2^n \cdot \#(\varphi_1) + \#(\varphi_2)$ with the default assignments of all variables set to 1 for one formula (and all 0s for the other formula). This uses n additional variables to scale by 2^n. This function is then simple enough that its inverse is computable in AC^0 to extract both counts and subtract to simulate gapP.

For #MON2SAT, f hides $\#(\varphi_2)$ since it is multiplied by $\#(\varphi_1)$. The power we are lacking here is the ability to enforce that variables are set to 0 and, thus, we are limited to all 1s for the default assignment. This makes it difficult to avoid multiplying both counts. Thus, we must use the circuit class TC^0 to extract both counts by performing integer division to simulate $[\#\mathrm{MON2SAT} - \#\mathrm{MON2SAT}]^{\log} = \#\mathrm{P}$. The results for the corresponding #2DNF fragments follow by the fact that these classes are closed under negation, see Lemma 27. Nevertheless, in Theorem 14 we show that #IMPL2DNF enriched with AC^0 postprocessing and #MON2DNF with TC^0 postprocessing already contains PH.

III. Discussion of Results and Outlook

In this paper, we present a new reduction from #SAT to #2SAT and #2DNF. Compared to the well-known reduction from Valiant, our reduction is direct and only requires two calls to a #2SAT (#2DNF) oracle. This

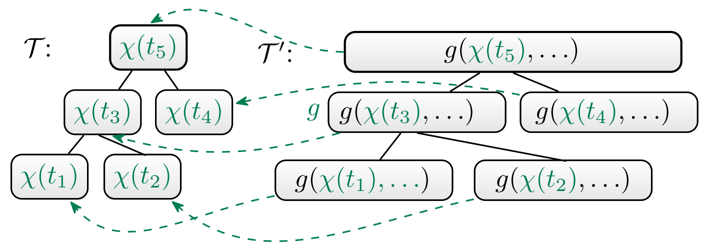


Fig. 3. Illustration of our reduction that is guided along any given tree decomposition $\mathcal{T}$ of the given formula. The reduction uses structural dependencies of $\mathcal{T}$ to ensure strong guarantees on the (tree)width of the resulting instance. $\mathcal{T}'$ is a tree decomposition of the resulting 2DNF formula(s).

reduction is not only conceptional simpler, but also computational: It can be carried out either in logarithmic space or in linear time. In particular, it also increases the size of the formula by at most a constant factor:

$$\#\mathrm{SAT} \in [\#2\mathrm{SAT} - \#2\mathrm{SAT}]^{\log} = [\#2\mathrm{DNF} - \#2\mathrm{DNF}]^{\log}.$$

It turns out, subtraction of two #2SAT or #2DNF calls is powerful enough to capture the larger class $\mathrm{gapP} \supsetneq \#\mathrm{P}$:

$$\mathrm{gapP} = [\#2\mathrm{SAT} - \#2\mathrm{SAT}]^{\log} = [\#2\mathrm{DNF} - \#2\mathrm{DNF}]^{\log},$$

which led to the title of the paper: Unless $\mathrm{NL} = \mathrm{NP}$, the class $\#\mathrm{P}$ is *strictly sandwiched* between one and two calls to a #2DNF oracle:

$$[\#2\mathrm{DNF}]^{\log} \subsetneq \#\mathrm{P} \subsetneq [\#2\mathrm{DNF} - \#2\mathrm{DNF}]^{\log}.$$

We also observe that the *subtraction* on the right side *is stronger than we thought* as it is enough to compensate for the absence of negation, i.e., the #2DNF formulas on the right side can be monotone. This "power", however, can be simulated by a single call to a #2DNF oracle if we allow a mild postprocessing via TC^0 circuits:

$$\#\mathrm{SAT} \in [\#\mathrm{MON2DNF}]^{\log}_{\mathrm{TC}^0},$$

which led to a new characterization of the polynomial hierarchy, i.e., a strengthening of Toda's Theorem:

$$\mathrm{PH} \subseteq [\#\mathrm{MON2DNF}]^{\log}_{\mathrm{TC}^0}.$$

As a further byproduct of our reduction, we also obtain a new algorithm that computes the count $\#(\varphi)$ in time $O\big(2^{\mathrm{itw}(\varphi)}|\varphi|\big)$, without the involved usage of zeta and Möbius transforms. The strong parameter-preservation guarantees of this reduction allowed us to establish *matching lower bounds* under (S)ETH, confirming that we can't expect significant improvements.

We would like to outline a few major directions for future work. The first path concerns further improvements on gapP. Since every function in spanL admits a *fully polynomial randomized approximation scheme* (FPRAS) [4], a natural question is in how far our work relates to the difference of calls to approximation algorithms? We also wonder about the (indirect) relationship of our reduction to the use of zeta and Möbius transforms in the recent algorithm by Slivovsky and Szeider [36]. Are there combinatorial or algebraic properties that our reduction indirectly encodes, which allow us to derive the result directly without the use of these techniques? Are there deeper algebraic connections (e.g., to group theory) leading to further insights into complexity?

The second direction is a better understanding of the exact power of a single #2SAT (#2DNF) call. From Theorems 12, 13, and 14, we know we only need one oracle call if we allow AC^0 or TC^0 postprocessing to capture $\#\mathrm{P}$, gapP, and PH. It would be interesting to precisely understand which circuits (i.e., how much postprocessing) are needed for each of the #SAT variants to, say, still capture PH. From the point of view of #SAT variants, we know $[\#2\mathrm{DNF}]^{\log} \subseteq \mathrm{spanL}$. On the other hand, the precise complexity of $[\#2\mathrm{SAT}]^{\log}$ remains unclear as we are unaware of any NL machine that can output a model of a 2SAT formula in a controlled and well-behaved way. For completeness of #2SAT we believe that we cannot achieve totP-hardness. What we can for sure claim is that such a reduction would require super logarithmic space. Since the decision version is NL-complete if we had a parsimonious reduction from any totP-complete problem, then the decision version for those problems can be solved in NL as well. However, problems in [3] have P-complete decision problems.

Observation 20. $[\#2\mathrm{SAT}]^{\log} \subsetneq \mathrm{totP}$ *assuming* $\mathrm{NL} \neq \mathrm{P}$.

Note that it is known that totP - totP and even $\mathrm{FP} - \mathrm{totP}$ [7] are equivalent to gapP, so one call to a totP-complete problem is enough to solve gapP. However, our tackled formalisms are expected to be significantly weaker than totP. For #MON2DNF we expect this might be even weaker than spanL. While we showed that $[\#\mathrm{MON2DNF} - \#\mathrm{MON2DNF}]^{\log}$ and $[\#\mathrm{MON2SAT} - \#\mathrm{MON2SAT}]^{\log}$ (and even more restricted versions) are enough to capture the hardness of gapP, we leave open whether replacement of one call by FP suffices. However, we believe not, as we expect the simulation of negation via subtraction to require symmetry. Unsurprisingly, the answer to these questions probably lies deep in circuit complexity, as we show that even slightly harder postprocessing is sufficient for only a single call (see Thm 12).

Conjecture 21. $[\mathrm{FP} - \#\mathrm{MONO2SAT}]$ *is strictly included in* $[\#\mathrm{MONO2SAT} - \#\mathrm{MONO2SAT}]$ *and* $[\#\mathrm{MON2DNF} - \#\mathrm{MON2DNF}]$.

To complete the picture (and to generate the Examples 23 and 26), we implemented the reductions presented within this paper via a first-order-like language that is then parsimoniously translated to #SAT. We tested these encodings using sophisticated state-of-the-art model counters [20]. On our tested examples, we can confirm that these systems are reasonably fast, even if there are billions of solutions (or numbers filling pages of paper). A thorough *practical* evaluation of our theoretical work, which covers all of the non-trivial implementation details of such a realization, is planned for a follow-up.

IV. Preliminaries

We consider proportional formulas in conjunctive normal form (CNFs) like $\varphi = (\neg a \vee b \vee c) \wedge (a \vee \neg b \vee c) \wedge (\neg c)$ as set-of-sets $\{\{\neg a, b, c\}, \{a, \neg b, c\}, \{\neg c\}\}$ and refer to its variables and clauses with $\mathrm{vars}(\varphi)$ and $\mathrm{clauses}(\varphi)$; respectively. We use the notation $\beta \subseteq \mathrm{vars}(\varphi)$ to refer to a subset β of the *variables* interpreted as those set to true. Such a set is called an *assignment*, and we say an assignment *satisfies* (is a *model* of) a clause $c \in \mathrm{clauses}(\varphi)$ if $\beta \cap c \neq \emptyset$ or $(\{x \mid \neg x \in c\}) \setminus \beta \neq \emptyset$. An assignment that satisfies *all* clauses of φ is a model of φ, which we denote by $\beta \models \varphi$. Finally, we denote the *number of models* of φ by $\#(\varphi)$, i.e., $\#(\varphi) := |\{ \beta \mid \beta \subseteq \mathrm{vars}(\varphi) \text{ and } \beta \models \varphi \}|$.

A. Fragments of Propositional Formulas

Every clause in 2CNF contains at most two literals, i.e., for two variables a, b the following clauses are allowed:

$$(a \vee b), \quad (\neg a \vee b), \quad (a \vee \neg b), \quad (\neg a \vee \neg b), \quad (a), \quad (\neg a), \quad (b), \quad (\neg b).$$

A HORN2CNF does not contain $(\neg a \vee \neg b)$, a MON2CNF only contains $(a \vee b)$ (i.e., no negation and no facts), and an IMPL2CNF does only contain $(\neg a \vee b) \equiv (a \rightarrow b)$ and $(a \vee \neg b) \equiv (b \rightarrow a)$, i.e., only positive implications. We make the same definitions for #DNFs, but instead refer to "IMPL2DNF" by 0,1-2DNF, as these are not implications. To be conform with the terminology used in the literature, we call counting problems over CNFs always #SAT (e.g., #IMPL2SAT) and over DNFs just #DNF (e.g., #0,1-2DNF).

B. Background in Structural Graph Theory

A *graph* G consists of a set of *vertices* $V(G)$ and a set of *edges* $E(G) \subseteq \binom{V(G)}{2}$. The *neighbors* of a vertex $v \in V(G)$ are $N(v) = \{ w \mid \{v, w\} \in E(G) \}$ and its *degree* is $|N(v)|$. This definition extends to vertex sets.

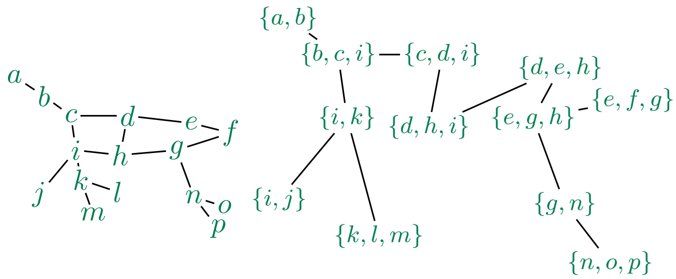
A *tree decomposition* (T, χ) of graph G consists of a rooted tree T and a mapping $\chi : V(T) \rightarrow 2^{V(G)}$ s.t.:

1) for every $v \in V(G)$ the set $\{ x \mid v \in \chi(x) \}$ is non-empty and connected in T;
2) for every $\{u, v\} \in E(G)$ there is at least one node $x \in V(T)$ with $\{u, v\} \subseteq \chi(x)$.

The *width* of a tree decomposition is the maximum size of its bags minus one, i. e., $\mathrm{width}(T, \chi) =$

$\max_{x \in V(T)} |\chi(x)| - 1$. The *treewidth* $\mathrm{tw}(G)$ of G is the minimum width among every decomposition of G. Then, $\mathrm{children}(t)$ is the set of child nodes of a node t in T.

Example 22. *The treewidth of the Ursa Major constellation (as graph shown on the left) is at most two, as proven by the tree decomposition on the right:*



Let $f : V(G) \to \{1, \ldots, |V(G)|\}$ be a *bijective mapping* that uniquely assigns a vertex to an integer from 1 to the number of vertices. The *dilation* of G and f is the maximum (absolute) difference between integers assigned to adjacent vertices, i.e., $\max_{\{u,v\} \in E(G)} |f(u) - f(v)|$. The *bandwidth* of G is the minimum dilation of G among every bijective mapping f for G.

C. Structure of Propositional Formulas

The *primal graph* of a CNF φ is the graph G_φ with $V(G_\varphi) = \mathrm{vars}(\varphi)$ that contains an edge between two vertices if the corresponding variables appear together in a clause. Parameters for formulas can be via the primal graph, e.g., $\mathrm{tw}(\varphi) := \mathrm{tw}(G_\varphi)$, $\mathrm{bw}(\varphi) := \mathrm{bw}(G_\varphi)s$.

Another representation is the *incidence graph* I_φ with $V(I_\varphi) = \mathrm{vars}(\varphi) \cup \mathrm{clauses}(\varphi)$ and $E(I_\varphi) = \{\, \{x, c\} \mid x \in \mathrm{vars}(\varphi), c \in \mathrm{clauses}(\varphi), \text{ and } \{x, \neg x\} \cap c \neq \emptyset \,\}$. This definition gives rise to *incidence parameters*, e.g., $\mathrm{itw}(\varphi) := \mathrm{tw}(I_\varphi)$, $\mathrm{ibw}(\varphi) := \mathrm{bw}(I_\varphi)$.

A *labeled* tree decomposition (T, χ, δ) of φ is a tree decomposition (T, χ) of G_φ, where every node gets assigned a set of labels using a labeling function $\delta \colon T \to 2^{\mathrm{clauses}(\varphi)}$. A labeled tree decomposition requires *(i)* for every node t of T and every $c \in \delta(t)$ that $\mathrm{vars}(c) \subseteq \chi(t)$ and *(ii)* $\mathrm{clauses}(\varphi) = \bigcup_{t \in V(T)} \delta(t)$. By introducing dummy nodes where necessary, we may assume without loss of generality that $|\delta(t)| \leq 1$ for all $t \in V(T)$.

V. Ingredients of the Main Lemma

In this section we discuss a new reduction from #SAT to #2SAT that increases the formula only linearly and preserves the input's treewidth up to an additive constant. Thereby we require two #2SAT oracle calls, followed by a subtraction, which establishes the main lemma.

Lemma 4 (Main Lemma)**.** *There is a* { *linear-time, logspace* } *algorithm mapping a* CNF φ *and a corresponding tree decomposition to* CNF*s* ψ_1, ψ_2 *with at most two variables per clause such that, for* $\rho \in \{\mathrm{tw}, \mathrm{itw}, \mathrm{bw}, \mathrm{ibw}\}$,[5]

$$\#(\varphi) = \#(\psi_1) - \#(\psi_2) \quad \textit{and}$$

$$\max\{\rho(\psi_1), \rho(\psi_2)\} \leq \alpha \cdot \rho(\varphi) + 14 \textit{ with } \alpha = 1.$$

For $\rho \in \{\mathrm{tw}, \mathrm{itw}\}$*, the resulting formulas can be restricted to the following fragments:*

(A) $\alpha = 1$ *and* ψ_1, ψ_2 *are* monotone*, i.e., do not contain negations; or*

(B) $\alpha = 3$ *and* ψ_1, ψ_2 *comprise binary* implications *and are* cubic *and* bipartite*, i.e., every variable occurs at most three times and the primal graph does not contain an odd cycle.*

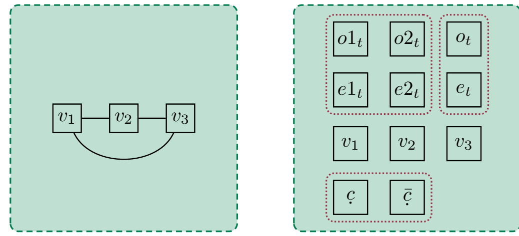


Fig. 4. (Left): A bag of a node t of a tree decomposition of the primal graph with 3 variables and a clause $c = v_1 \vee v_2 \vee v_3$ with $c \in \delta(t)$. (Right): Variables added to this bag by our reduction. Dotted boxes in red indicate that exactly one of these variables must be set to true in a model, otherwise we call the model rogue.

Let φ be a propositional formula and $\mathcal{T} = (T, \chi, \delta)$ be a labeled tree decomposition of G_φ. For the ease of presentation, we first show the case in which T is a path and every node (except the leaf) of T gets assigned a clause label, i.e., $\delta(t)$ is not empty for non-leaf nodes. Note that if one is not interested in structural properties of a tree decomposition, one could construct a trivial decomposition $\mathcal{T}$ in linear time, given *any ordering* among the clauses of φ. Indeed, such a $\mathcal{T}$ could then use a node for every clause, put every variable in every bag, and the labeling assigns every clause to its node.

An overview of our variables for a single node can be seen in Figure 4. We use variables[6] $v, \underset{\cdot}{c}, \bar{\underset{\cdot}{c}}$ for every variable $v \in \mathrm{vars}(\varphi)$ and every clause $c \in \mathrm{clauses}(\varphi)$. Further, we use auxiliary variables o_t and e_t for every node t in T to indicate that from the leaves of T up to node t, we assigned an even and odd number of clause variables to true, respectively. Additionally, we require auxiliary variables $o1_t, o2_t$ for o_t and $e1_t$, $e2_t$ for e_t. These auxiliary variables model auxiliary cases when defining o_t and e_t, which will be encoded symmetrically.

[5] Here $\mathrm{tw}(\varphi), \mathrm{itw}(\varphi), \mathrm{bw}(\varphi), \mathrm{ibw}(\varphi)$ denote treewidth and bandwidth of two different graphs associated with formula φ; see Section I-D.

[6] By $\underset{\cdot}{c}$ we just highlight the usage of c as a variable (symbol) and not refer to the object c itself.

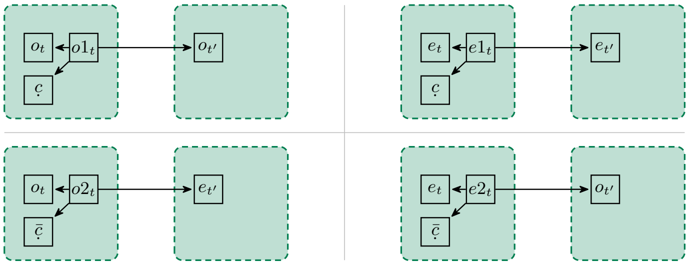


Fig. 5. An illustration of the four different *non-rogue cases* per tree decomposition node t (with child node t'), based on the single choice of variable in set $\{o1_t, o2_t, e1_t, e2_t\}$. Indeed, Equations (3)–(6) model all four potential cases, but also add many more *rogue models* we eliminate by subtraction. This works due to symmetry (see Definitions 24, 25).

Our reduction is similar to the inclusion-exclusion reduction from the introduction. For every clause $c = l_1 \vee \cdots \vee l_k$, we construct the following implications[7].

$$\underset{\cdot}{c} \rightarrow v \qquad \text{for every negative } l_i \in c, \text{i.e., } l_i = \neg v \quad (1)$$

$$v \rightarrow \overline{\underset{\cdot}{c}} \qquad \text{for every positive}^{7}\ l_i \in c, \text{i.e., } l_i = v \quad (2)$$

Intuitively, we guide the status of even (odd) along the tree decomposition. For a node t, we define four possible cases of being even or odd due to invalidating a clause or not, see Figure 5. So we construct clauses for every node t in T with $t' \in \text{children}(t)$ and $c \in \delta(t)$:

$$o1_t \rightarrow e_{t'} \quad o1_t \rightarrow \underset{\cdot}{c} \quad o1_t \rightarrow o_t \qquad \text{Case 1: odd by choosing } c \quad (3)$$

$$o2_t \rightarrow o_{t'} \quad o2_t \rightarrow \overline{\underset{\cdot}{c}} \quad o2_t \rightarrow o_t \qquad \text{Case 2: odd by not choosing } c \quad (4)$$

$$e1_t \rightarrow o_{t'} \quad e1_t \rightarrow \underset{\cdot}{c} \quad e1_t \rightarrow e_t \qquad \text{Case 1: even by choosing } c \quad (5)$$

$$e2_t \rightarrow e_{t'} \quad e2_t \rightarrow \overline{\underset{\cdot}{c}} \quad e2_t \rightarrow e_t \qquad \text{Case 2: even by not choosing } c \quad (6)$$

We use an additional auxiliary variable[8] x and add for every leaf node t in T:

$$x \rightarrow e_t \qquad \text{Initially, we choose 0 clauses (even)} \quad (7)$$

Example 23. *Recall our initial Example 19 and the running formula $\varphi = c_1 \wedge c_2 \wedge c_3$ with $c_1 = \neg a \vee b \vee c$, $c_2 = a \vee \neg b \vee c$, and $c_3 = \neg c$. Assume a labeled tree decomposition $\mathcal{T} = (T, \chi, \delta)$ of G_φ comprising the nodes t_0, t_1, t_2, t_3 such that $\delta(t_1) = c_1$, $\delta(t_2) = c_2$, and $\delta(t_3) = c_3$. Then, the reduction above constructs the following clauses, resulting in φ'.*

$$\underset{\cdot}{c_1} \rightarrow a \qquad \underset{\cdot}{c_2} \rightarrow b \qquad \underset{\cdot}{c_3} \rightarrow c \qquad (1)$$

$$b \rightarrow \overline{\underset{\cdot}{c_1}} \qquad c \rightarrow \overline{\underset{\cdot}{c_1}} \qquad a \rightarrow \overline{\underset{\cdot}{c_2}} \qquad c \rightarrow \overline{\underset{\cdot}{c_2}} \qquad (2)$$

$$\begin{array}{lll} o1_{t_1} \rightarrow e_{t_0} & o1_{t_1} \rightarrow \underset{\cdot}{c_1} & o1_{t_1} \rightarrow o_{t_1} \\ o1_{t_2} \rightarrow e_{t_1} & o1_{t_2} \rightarrow \underset{\cdot}{c_2} & o1_{t_2} \rightarrow o_{t_2} \\ o1_{t_3} \rightarrow e_{t_2} & o1_{t_3} \rightarrow \underset{\cdot}{c_3} & o1_{t_3} \rightarrow o_{t_3} \end{array} \qquad (3)$$

$$\begin{array}{lll} o2_{t_1} \rightarrow o_{t_0} & o2_{t_1} \rightarrow \overline{\underset{\cdot}{c_1}} & o2_{t_1} \rightarrow o_{t_1} \\ o2_{t_2} \rightarrow o_{t_1} & o2_{t_2} \rightarrow \overline{\underset{\cdot}{c_2}} & o2_{t_2} \rightarrow o_{t_2} \\ o2_{t_3} \rightarrow o_{t_2} & o2_{t_3} \rightarrow \overline{\underset{\cdot}{c_3}} & o2_{t_3} \rightarrow o_{t_3} \end{array} \qquad (4)$$

$$\begin{array}{lll} e1_{t_1} \rightarrow o_{t_0} & e1_{t_1} \rightarrow \underset{\cdot}{c_1} & e1_{t_1} \rightarrow e_{t_1} \\ e1_{t_2} \rightarrow o_{t_1} & e1_{t_2} \rightarrow \underset{\cdot}{c_2} & e1_{t_2} \rightarrow e_{t_2} \\ e1_{t_3} \rightarrow o_{t_2} & e1_{t_3} \rightarrow \underset{\cdot}{c_3} & e1_{t_3} \rightarrow e_{t_3} \end{array} \qquad (5)$$

$$\begin{array}{lll} e2_{t_1} \rightarrow e_{t_0} & e2_{t_1} \rightarrow \overline{\underset{\cdot}{c_1}} & e2_{t_1} \rightarrow e_{t_1} \\ e2_{t_2} \rightarrow e_{t_1} & e2_{t_2} \rightarrow \overline{\underset{\cdot}{c_2}} & e2_{t_2} \rightarrow e_{t_2} \\ e2_{t_3} \rightarrow e_{t_2} & e2_{t_3} \rightarrow \overline{\underset{\cdot}{c_3}} & e2_{t_3} \rightarrow e_{t_3} \end{array} \qquad (6)$$

$$x \rightarrow e_0 \qquad (7)$$

In order to count $\#(\varphi)$, we can compute $\#(\varphi' \cup \{x \rightarrow e_3\}) - \#(\varphi' \cup \{x \rightarrow o_3\}) = 204{,}452 - 204{,}450 = 2$. Note that it is not surprising that the constructed formulas admit a large number of models. Indeed, below, we will see that without the use of negation, there are even more satisfying assignments. Still, the reduction can be computed efficiently, and the key lies in the symmetrical construction and the use of subtraction.

Extension to Tree Decompositions: While the formula defined above already works for tree decompositions that are paths, for addressing tree decompositions[9] the following two cases are missing.

For tree decomposition nodes t with $\delta(t) = \emptyset$ (and $\text{children}(t) = \{t'\}$) we do not even use variables $o1_t, e1_t$ and only generate the following special case of Equations (4) and (6).

$$o2_t \rightarrow o_{t'} \qquad o2_t \rightarrow o_t \qquad e2_t \rightarrow e_{t'} \qquad e2_t \rightarrow e_t \quad (8)$$

In fact, the reduction can also be updated to accommodate so-called join nodes. For these tree decomposition nodes t with $\delta(t) = \emptyset$ (and $\text{children}(t) = \{t', t''\}$) we generate the following clauses, which are similar to Equations (3)–(6).

$$o1_t \rightarrow e_{t'} \quad o1_t \rightarrow o_{t''} \quad o1_t \rightarrow o_t \qquad \text{Case 1: odd by even/odd child nodes} \quad (9)$$

$$o2_t \rightarrow o_{t'} \quad o2_t \rightarrow e_{t''} \quad o2_t \rightarrow o_t \qquad \text{Case 2: odd by odd/even child nodes} \quad (10)$$

$$e1_t \rightarrow o_{t'} \quad e1_t \rightarrow o_{t''} \quad e1_t \rightarrow e_t \qquad \text{Case 1: even by odd/}$$

[7]We would like to express $\underset{\cdot}{c} \rightarrow \neg v$, which, unfortunately, is not an IMPL2CNF. However, recall that $v \rightarrow \overline{\underset{\cdot}{c}}$ is equivalent to its contraposition $\neg\overline{\underset{\cdot}{c}} \rightarrow \neg v$. While $\underset{\cdot}{c}$ and $\overline{\underset{\cdot}{c}}$ are different symbols, correctness is ensured by our definition of rogue models, see Definitions 24 and 25.

[8]Note that IMPL2CNF is a strict subset of HORN2CNF, only allowing implications of the form $a \rightarrow b$. Hence, we can't just add simple facts and need an additional auxiliary variable, which we refer to by x.

[9]We assume a tree decomposition using a binary tree (largest degree 3) such that degree-3 (join) nodes t have an empty labeling $\delta(t)$. Such a decomposition can be constructed in linear time in its size.

odd child nodes (11)

$e2_t \rightarrow e_{t'} \quad e2_t \rightarrow e_{t''} \quad e2_t \rightarrow e_t$ Case 2: even by even/ even child nodes (12)

Reduction R**.** Finally, let us denote with $R(\varphi, \mathcal{T})$ the formulas obtained by applying the above process, comprising Equations (1)–(12).

A. Solving #SAT by Subtracting Two #2SAT Calls

With the construction R from above, we can obtain the correct number of satisfying assignments via two calls to a #2SAT oracle, one to $\psi_1 := R(\varphi, \mathcal{T}) \cup \{x \rightarrow e_{\text{root}(T)}\}$ and one to $\psi_2 := R(\varphi, \mathcal{T}) \cup \{x \rightarrow o_{\text{root}(T)}\}$. The goal in the following is to prove that $\#(\varphi) = \#(\psi_1) - \#(\psi_2)$, requiring the central definition of *rogue models.*

Definition 24 (Rogue Model)**.** *Let t be a node in T. A model M of a formula $\varphi' \supseteq R(\varphi, \mathcal{T})$ is referred to by* rogue (at t) *whenever*

(i) $x \notin M$,
(ii) $|M \cap \{o1_t, o2_t, e1_t, e2_t\}| \neq 1$,
(iii) $|M \cap \{\underset{\cdot}{c}, \overline{c}\}| \neq 1$ *with* $c \in \delta(t)$*, or*
(iv) $|M \cap \{o_t, e_t\}| \neq 1$.

Intuitively, if there were zero rogue models, our reduction worked by the principle of inclusion-exclusion. Now recall Figure 2, which demonstrates the intuition that we need a bijection between rogue models of the first formula and those of the second formula. We rely on a construction that bijectively translates rogue models between ψ_1 and ψ_2. This aspect of symmetry for paths is visualized in Figure 6 (Top). The idea for constructing the symmetric model is to invert the parity of the rogue node closest to the root (and of all subsequent nodes, including the root). This immediately results in the corresponding symmetric rogue model, which preserves the rogue property of nodes. By construction, the symmetric rogue model of the symmetric rogue model is the rogue model itself (as desired).

The construction can also be generalized to trees T as visualized in Figure 6 (Bottom), where we just need to uniquely pick a path containing rogue models. Here it is fine to order all *root-to-leaf paths* of T and then pick the lexicographic smallest path containing a rogue model (which is unique). For the sake of concreteness, we thereby assume in Equations (9)–(12) that t' is always the child node that is on this path. In turn, we are left with a unique path, so the remaining construction proceeds similarly to the path case.

Formally, we define the construction of the symmetric rogue model as follows.

Definition 25 (Symmetric Rogue Model)**.** *Let M be a model of a formula φ' with $\varphi' \supseteq R(\varphi, \mathcal{T})$ that is rogue*

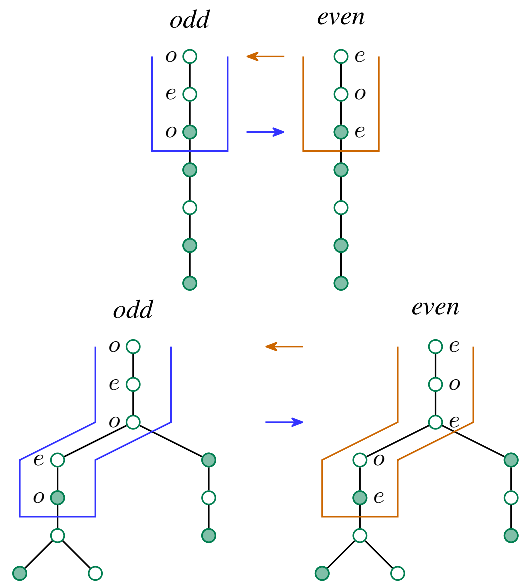


Fig. 6. Abstract visualization of the symmetric rogue model and its construction. Nodes of the tree T are given in green. Nodes filled in green indicate that the model is rogue at this filled node. The intuition is that if T is a path (Top), in the construction it suffices to change the parity of the rogue node closest to the root, which then also causes parity changes of remaining nodes upwards. These changes enable the transition from odd to even parity (and vice versa) and they are visualized in blue (and orange). Intuitively, the (first) rogue node enables a free change of parity, which is then propagated towards the root. In case T is not a path (Bottom), one can order root-to-leaf paths and just pick the lexicographic smallest path with a rogue node and continue similarly as in (Top).

at t. Assume that (1) there is no ancestor t' of t in T such that M is rogue at t', and that (2) t is on the lexicographic smallest root-to-leaf path in T. Then, the symmetric rogue model M' (of M) *is constructed as:*

- *If $x \notin M$, we define $M' = M$*
- *Otherwise, if $x \in M$:*
  - *Replace $o_t \in M$ by $e_t \in M'$ (and vice versa, i.e., $e_t \in M$ iff $o_t \in M'$).*
  - *For every ancestor t' of t in T, we replace $o_{t'} \in M$ by $e_{t'} \in M'$ and vice versa (i.e., $e_{t'} \in M$ iff $o_{t'} \in M'$), as well as $o1_{t'} \in M$ by $e1_{t'} \in M'$, $e1_{t'} \in M$ by $o1_{t'} \in M'$, $o2_{t'} \in M$ by $e2_{t'} \in M'$, $e2_{t'} \in M$ by $o2_{t'} \in M'$ (and vice versa)*
  - *If (a) either[10] $o_t \in M$ or $e_t \in M$, and (b) $|M \cap \{o1_t, o2_t, e1_t, e2_t\}| \geq 1$, we additionally replace $o1_t \in M$ by $e2_t \in M'$, $o2_t \in M$ by $e1_t \in M'$ and vice versa (i.e., $e1_t \in M$ by $o2_t \in M'$, $e2_t \in M$ by $o1_t \in M'$)*

We say that M is the symmetric rogue model *of M'.*

With this definition at hand, we commence with proving correctness of the reduction. To that end, we need to show that symmetric rogue models are well-defined, i.e., that the construction ensures that a symmetric rogue

[10]"Either ... or" refers to an exclusive disjunction.

model M' of a rogue model M is *(a)* a model and *(b)* rogue at a node t if and only M is rogue at t. This is established by Lemmas 28 and 29, where full proof details are given in Appendix 8. In Appendix 8-A we show that *structural parameters are linearly preserved.*

B. Reducing to Monotone Formulas

We reuse the same construction as in Equations (1)–(6), but in the following assume *fully labeled tree decompositions*, where also every variable is a label of a tree decomposition node. For a literal l over variable v, we let $inv(l)$ be the variable $\top_v$ if $l = \neg v$ and be $\bot_v$ otherwise. Intuitively these auxiliary variables are used to refer to the truth value for v. We update the inclusion-exclusion reduction such that for every clause $c = l_1 \vee l_2 \vee \cdots \vee l_k$ we construct *positive clauses*:

$$\underset{\cdot}{\overline{c}} \vee inv(l_1) \qquad \underset{\cdot}{\overline{c}} \vee inv(l_2) \qquad \ldots \quad \underset{\cdot}{\overline{c}} \vee inv(l_k) \tag{13}$$

Additionally, for every node t in T with $t' \in \text{children}(t)$ and $v \in \delta(t) \cap \text{vars}(\varphi)$, we add:

$$\overline{v} \vee \top_v \quad \overline{v} \vee \bot_v \quad \top_v \vee \bot_v \quad \text{Choosing } v \text{ sets } v \text{ to true and to false} \tag{14}$$

We slightly adapt Equations (3)–(6) such that for every node t in T with $t' \in \text{children}(t)$ and label $\alpha \in \delta(t)$, which can be either a clause or a variable, we construct:

$$\overline{o1_t} \vee e_{t'} \quad \overline{o1_t} \vee \alpha \quad \overline{o1_t} \vee o_t \quad \text{Case 1: odd by choosing } \alpha \tag{15}$$

$$\overline{o2_t} \vee o_{t'} \quad \overline{o2_t} \vee \overline{\alpha} \quad \overline{o2_t} \vee o_t \quad \text{Case 2: odd by not choosing } \alpha \tag{16}$$

$$\overline{e1_t} \vee o_{t'} \quad \overline{e1_t} \vee \alpha \quad \overline{e1_t} \vee e_t \quad \text{Case 1: even by choosing } \alpha \tag{17}$$

$$\overline{e2_t} \vee e_{t'} \quad \overline{e2_t} \vee \overline{\alpha} \quad \overline{e2_t} \vee e_t \quad \text{Case 2: even by not choosing } \alpha \tag{18}$$

By adapting Equation 7, we obtain:

$$\overline{x} \vee e_t \quad \text{Initially, we choose 0 clauses \& variables} \tag{19}$$

Example 26. *Recall the reduction given in Example 23. Assume a labeled tree decomposition $\mathcal{T} = (T, \chi, \delta)$ of G_φ comprising the nodes t_0, t_1, …, t_6 such that $\delta(t_1) = c_1$, $\delta(t_2) = c_2$, $\delta(t_3) = c_3$, $\delta(t_4) = a$, $\delta(t_5) = b$, and $\delta(t_6) = c$. Then, by constructing Equations (13)–(19) similarly to Example 23, we obtain a formula φ'. In order to count $\#(\varphi)$, we can compute $\#(\varphi' \cup \{\overline{x} \vee e_6\}) - \#(\varphi' \cup \{\overline{x} \vee o_6\}) = 2{,}110{,}863{,}758 - 2{,}110{,}863{,}756 = 2$.*

Note that as above, one can easily adapt this adaptation to the simpler types of tree decomposition nodes, see Equations (8)–(12). We refer to the adapted reduction comprising Equations (13)–(14), (15)–(18), and (19) by $R'(\varphi, \mathcal{T})$. Roughly, the idea is to introduce an additional type of label (for variables) and ensure that Equations (15)–(18) work for *both clause and variable labels*. By construction, a node of a labeled tree decomposition can only have one label (and therefore only one type). However, we do not care to manage these labels individually, but the idea is to keep track of the *parity of the combined number* of corresponding variables being true. Let us, as in the previous section, also stipulate $\psi'_1 := R'(\varphi, \mathcal{T}) \cup \{\overline{x} \vee e_{\text{root}(T)}\}$ and $\psi'_2 := R'(\varphi, \mathcal{T}) \cup \{\overline{x} \vee o_{\text{root}(T)}\}$.

Extensions of rogue models for *monotone formulas* and proofs are given in Appendix 8-B. Appendix 8-C generalizes the reduction to *cubic and bipartite formulas*.

VI. New Characterization of GapP

In this section, we show how Theorem 3 yields a more fine-grained characterization of gapP. Below we show how one can still model a *switch*, that enables us to change between satisfying assignments of one formula and to those of the other formula.

This switch construction has to be extended if we are only using monotone formulas (see Theorem 12).

Theorem 7 (Characterization of GapP)**.**

$$\begin{aligned}
&\text{gapP} \\
&= [\#2\text{SAT}{-}\#2\text{SAT}]^{\log} = [\#\text{IMPL}2\text{SAT}{-}\#\text{IMPL}2\text{SAT}]^{\log} \\
&= [\#0{,}1\text{-}2\text{DNF}{-}\#0{,}1\text{-}2\text{DNF}]^{\log} = [\#2\text{DNF}{-}\#2\text{DNF}]^{\log} \\
&= [\#\text{DNF}{-}\#\text{DNF}]^{\log} = [\#\text{MON}2\text{SAT}{-}\#\text{MON}2\text{SAT}]^{\log} \\
&= [\#\text{MON}2\text{DNF}{-}\#\text{MON}2\text{DNF}]^{\log} = \text{spanL}{-}\text{spanL}.
\end{aligned}$$

The characterization extends to cubic *and* bipartite *restrictions of* #IMPL2SAT *and* #0,1-2DNF*; even if both formulas use the same variables and differ by only one literal/variable occurrence.*

Proof. Proof details are given in Appendix 9. ☐

VII. A New Characterization of PH

Finally, we would like to give an outlook and some insights into many-one reductions that are enriched with additional postprocessing power on top of the resulting count. First, we observe the following.

Lemma 27. $[\#\text{MON}2\text{SAT}]^{\log}_{\text{TC}^0} = [\#\text{MON}2\text{DNF}]^{\log}_{\text{TC}^0}$,
$[\#\text{IMPL}2\text{SAT}]^{\log}_{\text{AC}^0} = [\#0{,}1\text{-}2\text{DNF}]^{\log}_{\text{AC}^0}$.

Proof. In both cases we observe that the classes under consideration are closed under inversion, that is,

$$[\#\text{MON}2\text{SAT}]^{\log}_{\text{TC}^0} = [2^n - \#\text{MON}2\text{DNF}]^{\log}_{\text{TC}^0} =$$

$$[\#\mathrm{MON2DNF}]^{\log}_{\mathrm{TC}^0} = [2^n - \#\mathrm{MON2SAT}]^{\log}_{\mathrm{TC}^0}.$$

Indeed, the subtraction for the inverse problem can be carried out in AC^0 and therefore the equations above hold. Analogously,

$$[\#\mathrm{IMPL2SAT}]^{\log}_{\mathrm{AC}^0} = [2^n - \#0,1\text{-}2\mathrm{DNF}]^{\log}_{\mathrm{AC}^0} = [\#0,1\text{-}2\mathrm{DNF}]^{\log}_{\mathrm{AC}^0} = [2^n - \#\mathrm{IMPL2SAT}]^{\log}_{\mathrm{AC}^0}. \quad \square$$

With this lemma and the reduction techniques from above, we obtain the following (proven in Appendix 10).

Theorem 12. $\mathrm{gapP} \subseteq [\#\mathrm{MON2SAT}]^{\log}_{\mathrm{TC}^0} = [\#\mathrm{MON2DNF}]^{\log}_{\mathrm{TC}^0}$.

Theorem 13. $\mathrm{gapP} \subseteq [\#\mathrm{IMPL2SAT}]^{\log}_{\mathrm{AC}^0} = [\#0,1\text{-}2\mathrm{DNF}]^{\log}_{\mathrm{AC}^0}$. *This statement holds even if* $\#\mathrm{IMPL2SAT}$ *is restricted to cubic and bipartite formulas.*

Now, we use both ingredients to establish a stronger characterization of PH.

Theorem 14 (Characterization of PH)**.**

$$\mathrm{PH} \subseteq [\#\mathrm{MON2SAT}]^{\log}_{\mathrm{TC}^0} = [\#\mathrm{MON2DNF}]^{\log}_{\mathrm{TC}^0} \subseteq \mathrm{P}^{\#2\mathrm{DNF}[1]}$$
$$\mathrm{PH} \subseteq [\#\mathrm{IMPL2SAT}]^{\log}_{\mathrm{AC}^0} = [\#0,1\text{-}2\mathrm{DNF}]^{\log}_{\mathrm{AC}^0} \subseteq \mathrm{P}^{\#2\mathrm{DNF}[1]}$$

References

[1] Eric Allender, Samir Datta, Andris Ambainis, David A. Mix Barrington, and Huong LêThanh. Bounded depth arithmetic circuits: Counting and closure. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languages and Programming*, pages 149–158, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[2] Carme Àlvarez and Birgit Jenner. A very hard log-space counting class. *Theor. Comput. Sci.*, 107(1):3–30, 1993. `doi:10.1016/0304-3975(93)90252-O`.

[3] Antonis Antonopoulos, Eleni Bakali, Aggeliki Chalki, Aris Pagourtzis, Petros Pantavos, and Stathis Zachos. Completeness, approximability and exponential time results for counting problems with easy decision version. *Theoretical Computer Science*, 915:55–73, 2022.

[4] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. #NFA admits an FPRAS: efficient enumeration, counting, and uniform generation for logspace classes. *J. ACM*, 68(6):48:1–48:40, 2021. `doi:10.1145/3477045`.

[5] V Arvind and Piyush P Kurur. Graph isomorphism is in spp. *Information and Computation*, 204(5):835–852, 2006.

[6] Eleni Bakali, Aggeliki Chalki, Andreas Göbel, Aris Pagourtzis, and Stathis Zachos. Guest column: A panorama of counting problems the decision version of which is in p3. *ACM SIGACT News*, 53(3):46–68, 2022.

[7] Eleni Bakali, Aggeliki Chalki, Sotiris Kanellopoulos, Aris Pagourtzis, and Stathis Zachos. On the power of counting the total number of computation paths of nptms. In *Annual Conference on Theory and Applications of Models of Computation*, pages 209–220. Springer, 2024.

[8] Eleni Bakali, Aggeliki Chalki, and Aris Pagourtzis. Characterizations and approximability of hard counting classes below. In *International Conference on Theory and Applications of Models of Computation*, pages 251–262. Springer, 2020.

[9] Richard Beigel, John Gill, and Ulrich Hertramp. Counting classes: Thresholds, parity, mods, and fewness. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 49–57. Springer, 1990.

[10] Richard Beigel, Nick Reingold, and Daniel A. Spielman. PP is closed under intersection. *J. Comput. Syst. Sci.*, 50(2):191–202, 1995. URL: https://doi.org/10.1006/jcss.1995.1017, `doi:10.1006/JCSS.1995.1017`.

[11] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability, Second Edition*. IOS Press, 2021.

[12] Hans L. Bodlaender, Paul S. Bonsma, and Daniel Lokshtanov. The fine details of fast dynamic programming over tree decompositions. In *8th International Symposium on Parameterized and Exact Computation (IPEC 2013)*, pages 41–53, 2013. `doi:10.1007/978-3-319-03898-8\_5`.

[13] Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, 1996.

[14] Hans L. Bodlaender, Arie M. C. A. Koster, Frank van den Eijkhof, and Linda C. van der Gaag. Pre-processing for triangulation of probabilistic networks. In *17th Conference in Uncertainty in Artificial Intelligence (UAI 2001)*, pages 32–39, 2001.

[15] Radu Curticapean. Block interpolation: A framework for tight exponential-time counting complexity. *Inf. Comput.*, 261:265–280, 2018. URL: https://doi.org/10.1016/j.ic.2018.02.008, `doi:10.1016/J.IC.2018.02.008`.

[16] Vilhelm Dahllöf, Peter Jonsson, and Magnus Wahlström. Counting models for 2SAT and 3SAT formulae. *Theor. Comput. Sci.*, 332(1-3):265–291, 2005. URL: https://doi.org/10.1016/j.tcs.2004.10.037, `doi:10.1016/J.TCS.2004.10.037`.

[17] Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlen. Exponential Time Complexity of the Permanent and the Tutte Polynomial. *ACM Trans. Algorithms*, 10(4):21:1–21:32, 2014. `doi:10.1145/2635812`.

[18] Arnaud Durand, Miki Hermann, and Phokion G Kolaitis. Subtractive reductions and complete problems for counting complexity classes. *Theoretical Computer Science*, 340(3):496–513, 2005.

[19] Stephen A. Fenner, Lance Fortnow, and Stuart A. Kurtz. Gap-definable counting classes. *J. Comput. Syst. Sci.*, 48(1):116–148, 1994. `doi:10.1016/S0022-0000(05)80024-8`.

[20] Johannes Klaus Fichte, Markus Hecher, and Florim Hamiti. The model counting competition 2020. *ACM J. Exp. Algorithmics*, 26:13:1–13:26, 2021. `doi:10.1145/3459080`.

[21] Jacob Focke, Dániel Marx, and Pawel Rzazewski. Counting List Homomorphisms from Graphs of Bounded Treewidth: Tight Complexity Bounds. *ACM Trans. Algorithms*, 20(2):11, 2024. `doi:10.1145/3640814`.

[22] Lance Fortnow and Nick Reingold. PP is closed under truth-table reductions. *Inf. Comput.*, 124(1):1–6, 1996. URL: https://doi.org/10.1006/inco.1996.0001, `doi:10.1006/INCO.1996.0001`.

[23] Johan Håstad. Almost optimal lower bounds for small depth circuits. In Juris Hartmanis, editor, *18th Annual ACM Symposium on Theory of Computing (STOC'86)*, pages 6–20. ACM, 1986. `doi:10.1145/12130.12132`.

[24] Ulrich Hertrampf. Relations among mod-classes. *Theoretical Computer Science*, 74(3):325–328, 1990.

[25] William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.*, 65(4):695–716, 2002. `doi:10.1016/S0022-0000(02)00025-9`.

[26] Thanh Minh Hoang and Thomas Thierauf. The complexity of the inertia and some closure properties of gapl. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005)*, pages 28–37. IEEE Computer Society, 2005. `doi:10.1109/CCC.2005.28`.

[27] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, 1988. `doi:10.1137/0217058`.

[28] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[29] Tuomas Laakkonen, Konstantinos Meichanetzidis, and John van de Wetering. Picturing counting reductions with the ZH-calculus. In *20th International Conference on Quantum Physics and Logic (QPL 2023)*, pages 89–113, 2023. `doi:10.4204/EPTCS.384.6`.

[30] Kuldeep S Meel, Sourav Chakraborty, and Umang Mathur. A faster fpras for# nfa. *Proceedings of the ACM on Management of Data*, 2(2):1–22, 2024.

[31] Mitsunori Ogihara, Thomas Thierauf, Seinosuke Toda, and Osamu Watanabe. On closure properties of #P in the context of PF ∘ #P. *J. Comput. Syst. Sci.*, 53(2):171–179, 1996. URL: https://doi.org/10.1006/jcss.1996.0059, `doi:10.1006/JCSS.1996.0059`.

[32] Mitsunori Ogiwara and Lane A. Hemachandra. A complexity theory for feasible closure properties. *J. Comput. Syst. Sci.*, 46(3):295–325, 1993. doi:10.1016/0022-0000(93)90006-I.

[33] Aris Pagourtzis and Stathis Zachos. The complexity of counting functions with easy decision version. In *International Symposium on Mathematical Foundations of Computer Science*, pages 741–752. Springer, 2006.

[34] Christos H Papadimitriou and Stathis K Zachos. Two remarks on the power of counting. In *Theoretical Computer Science: 6th Gl-Conference Dortmund, January 5–7, 1983*, pages 269–275. Springer, 1982.

[35] Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010. doi:10.1016/j.jda.2009.06.002.

[36] Friedrich Slivovsky and Stefan Szeider. A faster algorithm for propositional model counting parameterized by incidence treewidth. In *23rd International Conference on the Theory and Applications of Satisfiability Testing (SAT 2020)*, volume 12178 of *Lecture Notes in Computer Science*, pages 267–276. Springer, 2020. doi:10.1007/978-3-030-51825-7_19.

[37] Allan Sly. Computational transition at the uniqueness threshold. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 287–296. IEEE, 2010.

[38] Thomas Thierauf, Seinosuke Toda, and Osamu Watanabe. On closure properties of GapP. *Comput. Complex.*, 4:242–261, 1994. doi:10.1007/BF01206638.

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

[40] Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discret. Appl. Math.*, 8(1):85–89, 1984. doi:10.1016/0166-218X(84)90081-7.

[41] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

[42] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979. doi:10.1137/0208032.

[43] Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986. doi:10.1016/0304-3975(86)90135-0.

APPENDIX

8. PROOF OF THE MAIN LEMMA

Lemma 28 (Well-Definedness)**.** *Let φ be a* CNF*, $\mathcal{T}$ be a tree decomposition of it, and M be a satisfying assignment of ψ_1 that is rogue. The symmetric rogue model M' of M is a satisfying assignment of ψ_2. Vice versa, the result holds if roles of M and M' are swapped.*

Proof. Let M be rogue at a node t. Assume that (1) there is no indirect ancestor node of t (e.g., parent node) such that M is rogue at this node. So t is the node closest to the root of T with M being rogue at this node. Further, (2) t is on the lexicographic smallest root-to-leaf path in T. Let M' be the symmetric rogue model of M. We proceed by case distinction on *why* M is rogue at t.

Case *(i)*. We have $x \notin M$ and, therefore, M' is unique for M and vice versa.

In the following cases we have $x \in M$ and, thus, $e_{\text{root}(T)} \in M$. Furthermore, since t is also the first node such that M is rogue at t and M is a model of ψ_1, we conclude that $o_t \in M$ or $e_t \in M$. (Either $t = \text{root}(T)$ or M is not rogue at the parent node of t, requiring $o_t \in M$ or $e_t \in M$). Consequently, M can only be rogue at t due to property *(ii)*, *(iii)*, or *(iv)* in Definition 24 if both $o_t \in M$ and $e_t \in M$.

Case *(iv)*. Since both $o_t, e_t \in M$, the construction of M' results in M' being a model of ψ_2, i.e., the replacement of o_t by e_t (and vice versa) does not destroy model status of ψ_2.

Case *(iii)*. If $M \cap \{\underset{\cdot}{c}, \underset{\cdot}{\overline{c}}\} = \emptyset$, we have $M \cap \{o1_t, o2_t, e1_t, e2_t\} = \emptyset$. The replacement of $o_t \in M$ by $e_t \in M'$ (and vice versa) results in M' being a model of ψ_2. If $\underset{\cdot}{c}, \underset{\cdot}{\overline{c}} \in M$ and $|M \cap \{o1_t, o2_t, e1_t, e2_t\}| < 1$, again, it is easy to see that replacing $o_t \in M$ by $e_t \in M'$ (and vice versa) results in M' being a model of ψ_2. Otherwise, if $\underset{\cdot}{c}, \underset{\cdot}{\overline{c}} \in M$ and $|M \cap \{o1_t, o2_t, e1_t, e2_t\}| \geq 1$, note that M' is obtained from M by *additionally replacing* $o1_t \in M$ by $e2_t \in M'$ and $o2_t \in M$ by $e1_t \in M'$ (and vice versa, respectively). Since both $\underset{\cdot}{c}, \underset{\cdot}{\overline{c}} \in M$, this ensures that therefore M' is a model of ψ_2.

Case *(ii)*. If $M \cap \{o1_t, o2_t, e1_t, e2_t\} = \emptyset$, indeed, M' is a model of ψ_2 as well. Note that $|M \cap \{o1_t, o2_t, e1_t, e2_t\}| > 1$ can never occur, assuming we have neither Case *(iv)* nor Case *(iii)*. The roles of M and M' can be easily switched, as the replacements of Definition 25 are completely symmetric. □

Lemma 29 (Symmetry)**.** *Let φ be a* CNF*, $\mathcal{T}$ be a tree decomposition of it, and M be a satisfying assignment of ψ_1 that is rogue. Then, (I) a model M of ψ_1 is rogue at a node t iff the symmetric rogue model M' of ψ_2 is rogue at t (and vice versa with swapped M and M'); and, (II) mapping the rogue model of ψ_1 to its symmetric rogue model forms a bijection.*

Proof. We define a function f mapping models M of ψ_1 to corresponding symmetric rogue models M' of ψ_2. Suppose M is rogue at a node t' of T. In order to show that *(I)* $M' = f(M)$ is also rogue at t', let $t^\star$ be the node of T such that M is rogue at $t^\star$ with M not being rogue at an ancestor of $t^\star$. We distinguish the following cases:

Case t' is an ancestor of $t^\star$. By construction M is not rogue at $t^\star$ iff M' is not rogue at $t^\star$.

Case t' is a descendant of $t^\star$. Since M and M' are by construction identical regarding Definition 24 *(i)*–*(iv)*, we conclude that M is rogue at $t^\star$ iff M' is not rogue at $t^\star$.

Case $t' = t^\star$. Holds as the construction of Definition 25 does not change the rogue status of a model.

The proof works analogously if the roles of M and M' are swapped. It remains to show *(II)*: f is indeed a bijection. By Lemma 28, f is well-defined. Further, by the construction given in Definition 25, f is also injective. Suppose towards a contradiction that there was a rogue model M_2 of ψ_2 and two rogue models $M_1, M_1' \in f^{-1}(M_2)$. One can proceed by case distinction. Since we start replacing $o_{t'}$ by $e_{t'}$ (and vice versa) for nodes t' from $t^\star$ upwards in the direction towards the root of T, M_1 and M_1' coincide on the assignment of $o_{t'}$ and $e_{t'}$. The remaining interesting case is the last item of Definition 25. Observe that we only replace $o1_{t^\star}$ by $e2_{t^\star}$, $o2_{t^\star}$ by $e1_{t^\star}$ (and vice versa) if without this replacement the result is not a model. Therefore, $M_1 = M_1'$, which shows that f is injective.

It remains to show that f also is surjective. Suppose towards a contradiction that there is a rogue model M'' of ψ_2 such that $f^{-1}(M'')$ is not defined. However, we can construct a rogue model M''' of M'' according to Definition 25. Then, by Lemma 28 and (i) above, M''' is indeed a rogue model of ψ_1. This contradicts the assumption that $f^{-1}(M'')$ is not defined, since $M''' = f^{-1}(M'')$. □

Lemma 30. *(1) For every model M of ψ_1 or ψ_2 that is not rogue, $M \cap \text{vars}(\varphi)$ is an assignment over the variables of φ that invalidates at least $nc = |\{\underset{\cdot}{c} \mid c \in \text{clauses}(\varphi), \underset{\cdot}{c} \in M\}|$ clauses. (2) nc is odd iff M is a model of ψ_2.*

Proof. By construction, M invalidates at least nc clauses. Further, since M is not rogue, nc is odd iff M is a model of ψ_2. □

We now have all ingredients to prove Lemma 4, which requires us to show that *(a)* $\#(\varphi) = \#(\psi_1) - \#(\psi_2)$; *(b)* that $R(\varphi, \mathcal{T})$ can be computed in linear time or logspace; and *(c)* $\max\{\mathrm{tw}(\psi_1), \mathrm{tw}(\psi_2)\} \leq \mathrm{tw}(\varphi)$. The following three propositions establish these statements.

Proposition 31 (Correctness)**.** $\#(\varphi) = \#(\psi_1) - \#(\psi_2)$

Proof. As discussed in the introduction, the construction simulates the principle of inclusion-exclusion. We can count the number of models of φ as:

$$\#(\varphi) = 2^n - \sum_{\substack{M \subseteq 2^{\mathrm{vars}(\varphi)}, \exists c \in \mathrm{clauses}(\varphi),\\ M \not\models \{c\}}} 1 \quad + \sum_{\substack{M \subseteq 2^{\mathrm{vars}(\varphi)}, \exists c, c' \in \mathrm{clauses}(\varphi),\\ c \neq c', M \not\models c \vee c'}} 1 \quad + \cdots$$
$$+ \sum_{M \subseteq 2^{\mathrm{vars}(\varphi)}, \nexists c \in \mathrm{clauses}(\varphi), M \models \{c\}} (-1)^{|\varphi|},$$

which simplifies to
$$\sum_{\substack{M \subseteq 2^{\mathrm{vars}(\varphi)}, 0 \leq nc \leq |\varphi|,\\ M \text{ does not satisfy } \geq nc \text{ clauses in } \varphi}} (-1)^{nc}.$$

Therefore we can split this term into two parts, where we compute assignments that do not satisfy at least nc clauses with nc being even (ψ_1) and then subtract those assignments with nc being odd (ψ_2). As a result, the goal is to encode the result of exclusive-or, i.e., whether an assignment does not satisfy at least nc clauses with nc being even (odd). We refer to those assignments where nc is even by

$$E = \{M \subseteq 2^{\mathrm{vars}(\varphi)} \mid 0 \leq nc \leq |\varphi|, M \text{ does not satisfy } \geq nc \text{ clauses in } \varphi, nc \equiv 0 (\mathrm{mod} 2)\}.$$

The assignments with nc being odd are referred to by

$$O = \{M \subseteq 2^{\mathrm{vars}(\varphi)} \mid 0 \leq nc \leq |\varphi|, M \text{ does not satisfy } \geq nc \text{ clauses in } \varphi, nc \equiv 1 (\mathrm{mod} 2)\}.$$

By Lemma 30, every assignment in E can be extended to a satisfying assignment of ψ_1, i.e., $|E| \leq \#(\psi_1)$. Analogously, $|O| \leq \#(\psi_2)$. However, by Lemma 29, there is a bijective function between the set R of rogue models of ψ_1 and the rogue models of ψ_2. Consequently,

$$\#(\varphi) = \big(|E| + |R|\big) - \big(|O| + |R|\big) = \#(\psi_1) - \#(\psi_2). \quad \square$$

Proposition 32. *The reduction $R(\varphi, \mathcal{T})$ can be computed in linear time or logarithmic space for a given tree decomposition. We have $|\psi_1| + |\psi_2| \in O\big(|\varphi| + |\mathcal{T}|\big)$.*

Proof. Let $\mathcal{T} = (T, \chi, \delta)$ be the given labeled TD. Without loss of generality, we may assume that the size of $\mathcal{T}$ is linear in $|\varphi|$ [13]. This still holds for labeled TD $\mathcal{T}$, which only linearly increases the size in the worst case (compared to an unlabeled TD). Then, Equation (1), (2) is computed for every clause $c \in \mathrm{clauses}(\varphi)$ and Equations (3)–(12) are computed for every node in T and in the context of a single clause. Consequently, it is easy to see that the size is linearly bounded for both ψ_1 and ψ_2. The logspace bound follows, as we only need a constant number of pointers to the input. $\square$

A. Preserving Structural Parameters

Let φ be a CNF formula and recall R, as well as ψ_1 and ψ_2 from above. We are ready to show our strong guarantees for structural parameters.

Lemma 33. $\max\{\mathrm{tw}(\psi_1), \mathrm{tw}(\psi_2)\} \leq \mathrm{tw}(\varphi) + 13$ *and* $\max\{\mathrm{itw}(\psi_1), \mathrm{itw}(\psi_2)\} \leq \mathrm{itw}(\varphi) + 14$.

Proof. Take any tree decomposition $\mathcal{T} = (T, \chi)$ of G_φ (of width $\mathrm{tw}(\varphi)+1$). Without loss of generality, we may assume that every node in T has at most 2 child nodes. Indeed, this comes with a factor $\mathrm{tw}(\varphi)$ overhead in the number of nodes by just adding intermediate copies of nodes. From this, we can easily obtain a labeled TD $\mathcal{T}' = (T', \chi', \delta)$ of width $\mathrm{tw}(\varphi) + 1$.

Then, we obtain a tree decomposition $\mathcal{T}'' = (T', \chi'')$ of both G_{ψ_1} and G_{ψ_2}, where we define χ'' as follows. For every node t of T'', let $\chi''(t) = \chi'(t) \cup \{x, o1_t, o2_t, e1_t, e2_t, e_t, o_t\} \cup \{e_{t'}, o_{t'} \mid t' \in \mathrm{children}(t)\} \cup \{c, \overline{c} \mid c \in \delta(t)\}$. It is therefore easy to see that $|\chi''(t)| \leq |\chi'(t)| + (9 + 2 \cdot |\mathrm{children}(t)|) = |\chi'(t)| + 13$, since $|\mathrm{children}(t)| \leq 2$.

For the second claim, suppose that $\mathcal{T}'$ was a tree decomposition of the incidence graph I_φ. Then, we can reuse the same construction as above, to convert the resulting tree decomposition $\mathcal{T}''$ into a tree decomposition $\mathcal{T}'''$ of both I_{ψ_1} and I_{ψ_2}. Indeed, we additionally need to add vertices for the clauses in I_{ψ_1} (I_{ψ_2}) we generated in Equations (1)–(7). However, for each bag it suffices to add one of these vertices at a time (and just duplicate the bag several times). This results in a tree decomposition $\mathcal{T}'''$ of I_{ψ_1} and I_{ψ_2}, where the bag sizes are just increased by 1, establishing the desired claim. Note that due to the resulting chain of (copied) bags, the resulting tree decomposition might just be of linear size in the instance size (as each variable occurrence in a clause is treated at most once). $\square$

Observe that the same argument immediately applies to pathwidth, where every TD node has at most one child node, so we obtain even $+12$ instead of $+14$. For the bandwidth parameter, we obtain similar results, which yield tight lower bounds (see Theorem 16).

Lemma 34. *We can slightly modify ψ_1 and ψ_2 such that the following claim holds:*

$$\max\{\mathrm{bw}(\psi_1), \mathrm{bw}(\psi_2)\} \leq \mathrm{bw}(\varphi) + 11 \textit{ and}$$

$$\max\{\mathrm{ibw}(\psi_1), \mathrm{ibw}(\psi_2)\} \leq \mathrm{ibw}(\varphi){+}12.$$

Proof. Take a bijective mapping f of G_φ with $n = |\operatorname{vars}(\varphi)|$. In this proof we will process f in batches of size k, where the ith batch ($0 \leq i < b$ with $b = \lfloor \frac{n}{k} \rfloor$) ranges from index ik to index $i(k{+}1){-}1$. The variables (vertices) of the ith batch are addressed by $B_i = \{v_{ik}, \ldots, v_{i(k+1)-1}\}$. Consequently, we can simulate a tree decomposition $\mathcal{T}_f$ of G_φ, which we can pass to R. This decomposition is a path where the ith bag comprises elements of the ith batch as well as the $(i-1)$st batch (if it exists). Hence, the width of $\mathcal{T}_f$ is $2k$.

However, for bandwidth, the construction of ψ_1 and ψ_2 needs to be slightly adapted as follows. Instead of variable x, we use copies $x_0, \ldots, x_{b-1}$ and replace x in Equation (7) by x_{b-1} (and every other occurrence of x is replaced by x_0). Further, we add implications $x_0 \rightarrow x_1$, $x_0 \leftarrow x_1$, $\ldots$, $x_{b-2} \rightarrow x_{b-1}$, $x_{b-2} \leftarrow x_{b-1}$.

Then, similarly to Lemma 33 above, we can look at clauses in φ one by one. If, say, there is more than one clause over variables contained in the ith batch, we could just have created an intermediate batch between the ith and the $(i+1)$st batch. This batch is over copy variables $v'_{ik}, \ldots, v'_{i(k+1)-1}$ and we would have constructed clauses (implications) for every index j in this batch: $v_j \rightarrow v'_j$ and $v_j \rightarrow v'_j$ to φ (and renamed occurrences of v_j in φ involving variables in later batches following i).

Hence, we can construct a modified bijective mapping f' of both G_{φ_1} and G_{φ_2}, using similar constructions as in Lemma 33. Thereby, analogously to above, every batch i gets extended by $\{x_i, o1_i, o2_i, e1_i, e2_i, e_i, o_i\} \cup \{e_{i-1}, o_{i-1} \mid i-1 > 0\} \cup \{\underset{\cdot}{c}, \overline{\underset{\cdot}{c}} \mid c \in \operatorname{clauses}(\varphi)$ considered inbatch $i\}$. Observe that this establishes the first claim, as f' is a bijective mapping of dilation at most $\mathrm{bw}(\varphi) + 11$.

The argument for incidence bandwidth works similarly to Lemma 33 above, thereby increasing from $+11$ to $+12$. However, in addition, we need to add intermediate copies of batches as demonstrated above. $\square$

B. Proof for Monotone Formulas

We are ready to extend our definition of a the rogue model (Definition 24) for monotony below. Note that Equation (14) introduces another reason, *why a model can be rogue* at a node. This is formalized in Definition 35 by the added item (iiib). As in Section V-A, by establishing a bijection we then utilize the *power of subtraction* to eliminate rogue models.

Definition 35 (Rogue Model for Monotony)**.** *Let φ be a* CNF, *$\mathcal{T} = (T, \chi, \delta)$ be a fully labeled TD of φ, and t be a node in T. Then, a model M of a formula $\varphi' \supseteq R'(\varphi, \mathcal{T})$ is referred to by* rogue (at t) *whenever*

(i) $\overline{x} \in M$,

(ii) $|M \cap \{\overline{o1_t}, \overline{o2_t}, \overline{e1_t}, \overline{e2_t}\}| \neq |\{\overline{o1_t}, \overline{o2_t}, \overline{e1_t}, \overline{e2_t}\} \cap \operatorname{vars}(\varphi')| - 1$,

(iii) $|M \cap \{\alpha, \overline{\alpha}\}| \neq 1$ *with* $\alpha \in \delta(t)$,

(iiib) $\{\top_v, \bot_v\} \subseteq M$ *with* $v \in \delta(t) \cap \operatorname{vars}(\varphi)$, *or*

(iv) $|M \cap \{o_t, e_t\}| \neq 1$.

Condition (ii) usually means $|M \cap \{\overline{o1_t}, \overline{o2_t}, \overline{e1_t}, \overline{e2_t}\}| \neq 3$. The construction of the symmetric rogue model works analogously as in Definition 25, where instead of x we use $\overline{x}$ and instead of $o1_t$, $o2_t$, $e1_t$, $e2_t$ we put $\overline{o1_t}$, $\overline{o2_t}$, $\overline{e1_t}$, $\overline{e2_t}$, respectively.

Definition 36 (Symmetric Rogue Model for Monotony)**.** *Let M be a model of a formula φ' with $\varphi' \supseteq R(\varphi, \mathcal{T})$ that is rogue at t with $\alpha \in \delta(t)$. Assume that (1) there is no ancestor t' of t in T such that M is rogue at t' and that (2) t is on the lexicographic smallest root-to-leaf path of T. The* symmetric rogue model M' (of M) *is constructed by:*

- *If $\overline{x} \in M$, we define $M' = M$*
- *Otherwise, if $\overline{x} \notin M$:*
  - *Replace $o_t \in M$ by $e_t \in M'$ (and vice versa, i.e. $e_t \in M$ iff $o_t \in M'$).*
  - *For every ancestor t' of t in T, we replace $o_{t'} \in M$ by $e_{t'} \in M'$ and vice versa (i.e. $e_{t'} \in M$ iff $o_{t'} \in M'$), as well as $\overline{o1_{t'}} \notin M$ by $\overline{e1_{t'}} \notin M'$, $\overline{e1_{t'}} \notin M$ by $\overline{o1_{t'}} \notin M'$, $\overline{o2_{t'}} \notin M$ by $\overline{e2_{t'}} \notin M'$, $\overline{e2_{t'}} \notin M$ by $\overline{o2_{t'}} \notin M'$ (and vice versa)*
  - *If (a) either $o_t \in M$ or $e_t \in M$, and (b) $|M \cap \{\overline{o1_t}, \overline{o2_t}, \overline{e1_t}, \overline{e2_t}\}| \leq 3$, we additionally replace $\overline{o1_t} \notin M$ by $\overline{e2_t} \notin M'$, $\overline{o2_t} \notin M$ by $\overline{e1_t} \notin M'$ and vice versa (i.e. $\overline{e1_t} \notin M$ by $\overline{o2_t} \notin M'$, $\overline{e2_t} \notin M$ by $\overline{o1_t} \notin M'$)*
  - *If (a), (b), and (c) $\{\top_\alpha, \bot_\alpha\} \subseteq M$, we additionally replace $\alpha \in M$ by $\overline{\alpha} \in M'$ and vice versa (i.e. $\overline{\alpha} \in M$ by $\alpha \in M'$)*

We say M is the symmetric rogue model *of M'.*

Lemma 37. *Let φ be a* CNF, *$\mathcal{T}$ be a tree decomposition of it, and M be a satisfying assignment of ψ'_1 that is rogue. Then, the symmetric rogue model M' of M is a satisfying assignment of ψ'_2. Vice versa, the result holds if roles of M and M' are swapped.*

Proof. Let M be rogue at a node t and assume that (1) there is no indirect ancestor node of t (e.g., parent node) such that M is rogue at this node and that (2) t is on the lexicographic smallest root-to-leaf path of T. So t is the node closest to the root of T with M being rogue at this node. Let M' be the symmetric rogue model of M.

The proof works analogously to Lemma 28. The only missing case is where M is rogue at t only due (iiib) of Definition 35. Then we simply perform the same replacements, as in Case (iii). Since these replacements require that if $\alpha \in M$ then $\overline{\alpha} \in M'$ (and vice versa), the last item of Definition 36 ensures that M' is a model of φ_2'. As for Lemma 28, the roles of M and M' can be switched by symmetry. $\square$

Lemma 38. *Let φ be a* CNF*, $\mathcal{T}$ be a fully labeled tree decomposition of it, and M be a satisfying assignment of ψ_1' that is rogue. Then, (I) a model M of ψ_1' is rogue at a node t iff the symmetric rogue model M' of ψ_2' is rogue at t (and vice versa with swapped M and M'); and (II) mapping the rogue model of ψ_2' to its symmetric rogue model forms a bijection.*

Proof. We define a function f by mapping a model M of ψ_1' to its corresponding symmetric rogue model M' of ψ_2'. Suppose M is rogue at a node t' of T. In order to show that *(I)* $M' = f(M)$ is also rogue at t', let $t^\star$ be the node of T such that M is rogue at $t^\star$ with M not being rogue at an ancestor of $t^\star$. We distinguish the following cases.

Case t' is an ancestor of $t^\star$. By construction M is not rogue at $t^\star$ iff M' is not rogue at $t^\star$.

Case t' is a descendant of $t^\star$.
Since M and M' are by construction identical regarding Definition 35 *(i)–(iv)*, we follow that M is rogue at $t^\star$ iff M' is not rogue at $t^\star$.

Case $t' = t^\star$.
Holds since the construction of Definition 36 does not change the rogue status of a model.

The proof works analogously if the roles of M and M' are swapped. It remains to show that *(II)* f is indeed a bijection. By Lemma 37, f is well-defined. Further, by the construction given in Definition 36, f is also injective. Indeed, suppose towards a contradiction that there was a rogue model M_2 of ψ_2' and two rogue models $M_1, M_1' \in f^{-1}(M_2)$. One can proceed by case distinction. Since we start replacing $o_{t'}$ by $e_{t'}$ (and vice versa) for nodes t' from $t^\star$ upwards in the direction towards the root of T, M_1 and M_1' coincide on the assignment of $o_{t'}$ and $e_{t'}$. The remaining interesting case is the last item of Definition 36. Observe that we only replace α by $\overline{\alpha}$ (and vice versa) if without this replacement the result is not a model. Indeed, if $\{\alpha, \overline{\alpha}\} \subseteq M$, the replacement does not achieve anything. Otherwise, *(a)* and *(b)* of Definition 36 implies that $|M \cap \{\overline{o1_t}, \overline{o2_t}, \overline{e1_t}, \overline{e2_t}\}| = 3$. Consequently, M is rogue at $t^\star$, only due to (iiib) of Definition 35, i.e. the replacement of α ($\overline{\alpha}$) in M is required for the resulting assignment to be a model of Equation (14). Therefore, $M_1 = M_1'$, which shows that f is injective.

Analogously to Lemma 29, f is surjective. $\square$

Clearly, ψ_1' and ψ_2' are *monotone* formulas in 2CNF. It is easy to see that the modifications from $R'(\varphi, \mathcal{T})$ compared to $R(\varphi, \mathcal{T})$ do not increase the treewidth and increase the size of the produced formulas only by a constant factor. By utilizing Lemma 37 and Lemma 38 as in the previous section, we can conclude:

Corollary 39. *There is a linear-time algorithm that maps formulas φ to formulas ψ_1 and ψ_2* ***without negation*** *and with at most two variables per clause such that*

$$\#(\varphi) = \#(\psi_1) - \#(\psi_2) \quad \textit{and}$$
$$\max\{\mathrm{tw}(\psi_1), \mathrm{tw}(\psi_2)\} \leq \mathrm{tw}(\varphi) + 13,$$
$$\max\{\mathrm{itw}(\psi_1), \mathrm{itw}(\psi_2)\} \leq \mathrm{itw}(\varphi) + 14.$$

C. Reducing to Cubic and Bipartite Formulas

Before we discuss stronger results by restricting #IMPL2SAT to formulas of degree at most 3 and bipartite primal graphs, we briefly mention the following.

Proposition 40. *There is a linear-time conversion from a formula φ in* CNF *to a formula φ' in* 3CNF*, such that the* $\mathrm{tw}(\varphi') \leq \mathrm{tw}(\varphi)+2$. *If additionally, every variable in φ' occurs at most 3 times (not of the same sign), we still obtain* $\mathrm{tw}(\varphi') \leq 3\,\mathrm{tw}(\varphi)+2$, $\mathrm{itw}(\varphi') \leq 3\,\mathrm{itw}(\varphi)+3$.

Proof. The first claim can be easily established by taking any labeled tree decomposition $\mathcal{T} = (T, \chi, \delta)$ of G_φ (of width $\mathrm{tw}(\varphi) + 1$). Then, for every node t in T, we split up long clauses $c = l_1 \vee l_2 \vee \cdots \vee l_k$ such that $c \in \delta(t)$, via auxiliary variables $a_1, \ldots, a_{k-1}$ and by constructing auxiliary clauses of the form $l_1 \vee l_2 \vee a_1$, $\neg a_1 \vee l_3 \vee a_2$, $\neg a_2 \vee l_4 \vee a_3$, $\ldots$, $\neg a_{k-1} \vee l_k$. We refer to the resulting formula containing every auxiliary clause by φ'. It is easy to see how we obtain a tree decomposition $\mathcal{T}'$ of φ' from $\mathcal{T}$. We take $\mathcal{T}$ and basically duplicate nodes (i.e., we replace a node in T by a path, similarly to Lemma 33) and add to each duplicate bag at most two auxiliary variables a_i, a_{i+1}. Then, as $|\delta(t)| \leq 1$, the width of the resulting tree decomposition $\mathcal{T}'$ is bounded by $\mathrm{tw}(\varphi) + 1 + 2$. Indeed, this can be done such that all variables of every constructed auxiliary clause are covered by $\mathcal{T}'$, i.e., $\mathcal{T}'$ is a tree decomposition of $G_{\varphi'}$.

If instead $\mathcal{T}$ were a tree decomposition of I_φ of width $\mathrm{itw}(\varphi) + 1$, we could still apply the same idea as above, but the labeling δ is insufficient since clauses span over several bags. Consequently, when we guide the construction of auxiliary variables along $\mathcal{T}$, we require to add for *each* clause vertex c in a bag the corresponding auxiliary variable a_i used in the previous bag(s). After a_i is not used anymore, we could duplicate the bag and add a_{i+1} to the fresh bag. It is therefore easy to see that this, unfortunately, causes a factor 2: $\mathrm{itw}(\varphi') \leq 2\,\mathrm{itw}(\varphi)$, as there can be up to $\mathrm{itw}(\varphi)$ clause vertices in a bag (each of these might need to keep an auxiliary vertex a_i).

For the second claim, we can ensure that a variable occurs at most 3 times in a clause, using established techniques [40]. Thereby, we create a copy v_i for every variable appearance v and chain these, e.g., $v \rightarrow v_2, \ldots, v_u \rightarrow v$. However, while the creation of such a (chain of) implications can be guided along a tree decomposition $\mathcal{T}$ (similar to above), in the worst case this requires that for every element in the bag, we also need to add the directly preceding copy variable form the previous bag, as well as the first copy variable to the bag (which we need for "closing" the cycle). Unfortunately, this already causes a factor 3 (worst-case) overhead: $\mathrm{tw}(\varphi') \leq 3\,\mathrm{tw}(\varphi)$, but it is easy to see that this can be combined with splitting clauses from above. Further, we may assume that not all of occurrences of each variable are of the same sign. If they were, we could combine the previous step of copying variables to remove those: Suppose a variable x occurs 3 times in the form of the same literal l. Then we replace the three occurrences of l by literals l, l_2, l_3 of the same sign as l, but over variables x, x_2, x_3. Then we add clauses $l \rightarrow l_2$, $l_2 \rightarrow l_3$, $l_3 \rightarrow l$, which ensures equivalence. Observe that this can be carried out with an overhead of $+2$, as we can do this for each variable x independently by copying bags. This results in $\mathrm{tw}(\varphi') \leq 3\,\mathrm{tw}(\varphi) + 2$. It is easy to see that then we obtain $\mathrm{itw}(\varphi') \leq 3\,\mathrm{itw}(\varphi) + 3$ since we might need to add vertices for constructed clauses one-by-one. □

By Proposition 40, we may assume a formula φ in 3CNF, where every variable occurs at most 3 times, but not with a single sign. Observe that the formula ψ constructed by Equations (1)–(6) on φ and a labeled tree decomposition $\mathcal{T} = (T, \chi, \delta)$ is already bipartite. Indeed, edges only occur between sets $U = \{v, e1_t, e2_t, o1_t, o2_t, x \mid v \in \mathrm{vars}(\varphi), t \text{ in } T\}$ and $V = \{\underset{\cdot}{c}, e_t, o_t \mid c \in \mathrm{clauses}(\varphi), t \text{ in } T\}$.

To preserve this bipartite property and ensure maximum degree 3, we need to update $R(\varphi, \mathcal{T})$ by adding additional clauses. For each clause $c \in \mathrm{clauses}(\varphi)$, we add additional auxiliary variables c', c'', $\overline{c}'$, and $\overline{c}''$ and construct the following clauses.

$$\underset{\cdot}{c} \rightarrow c' \qquad c' \rightarrow c'' \qquad\qquad \underset{\cdot}{\overline{c}} \rightarrow \overline{c}' \qquad \overline{c}' \rightarrow \overline{c}'' \tag{20}$$

Further, we replace every occurrence of $\underset{\cdot}{c}$ and $\underset{\cdot}{\overline{c}}$ in Equations (1) and (2) by c'' and $\overline{c}''$, respectively. Observe that this requires to add an additional condition to Definition 24, which is fulfilled if one of the copies of $\underset{\cdot}{c}$ ($\underset{\cdot}{\overline{c}}$) are assigned differently. Indeed, it could be that in a satisfying assignment, e.g., c'' is assigned to 1, but c' is not. To accommodate this, we update Definition 25 on symmetric rogue models, as outlined below.

Analogously, for every non-root node t^* in T, we add auxiliary variables e'_{t^*}, e''_{t^*}, o'_{t^*}, o''_{t^*} and construct:

$$e_{t^*} \rightarrow e'_{t^*} \quad e'_{t^*} \rightarrow e''_{t^*} \qquad o_{t^*} \rightarrow o'_{t^*} \quad o'_{t^*} \rightarrow o''_{t^*} \tag{21}$$

Then, it remains to replace in Equations (3)–(6) those occurrences of $e_{t'}$ and $o_{t'}$ where $t' = t^*$, by e''_{t^*} and o''_{t^*}, respectively. We refer to the adapted Reduction by $R(\varphi, \mathcal{T})$.

In turn, these additional clauses not only preserve the bipartite property, but they also ensure maximum degree 3. We refer by $R^{\text{C+B}}(\varphi, \mathcal{T})$ to the reduction obtained from modifying $R(\varphi, \mathcal{T})$ as outlined above. In order to show correctness for the reduction similar to Lemmas 28–30, we require an updated definition of the (symmetric) rogue model below. As above, $\psi_1^{\text{C+B}} := R^{\text{C+B}}(\varphi, \mathcal{T}) \cup \{x \rightarrow e_{\mathrm{root}(T)}\}$ and $\psi_2^{\text{C+B}} := R^{\text{C+B}}(\varphi, \mathcal{T}) \cup \{x \rightarrow o_{\mathrm{root}(T)}\}$.

Definition 41 (Rogue Model For Cubic and Bipartite)**.** *Let t be a node in T. A model M of a formula $\varphi' \supseteq R^{\text{C+B}}(\varphi, \mathcal{T})$ is referred to by* rogue (at t) *whenever*

(i) $x \notin M$,
(ii) $|M \cap \{o1_t, o2_t, e1_t, e2_t\}| \neq 1$,
(iii) $|M \cap \{\underset{\cdot}{c}, \underset{\cdot}{\overline{c}}\}| \neq 1$, $|M \cap \{\underset{\cdot}{c}, c', c''\}| \notin \{0, 3\}$, *or* $|M \cap \{\underset{\cdot}{\overline{c}}, \overline{c}', \overline{c}''\}| \notin \{0, 3\}$ *with* $c \in \delta(t)$, *or*
(iv) $|M \cap \{o_t, e_t\}| \neq 1$, $|M \cap \{o_t, o'_t, o''_t\}| \notin \{0, 3\}$, *or* $|M \cap \{e_t, e'_t, e''_t\}| \notin \{0, 3\}$.

Then, we can still bijectively translate rogue models between $\psi_1^{\text{C+B}}$ and $\psi_2^{\text{C+B}}$.

Definition 42 (Symmetric Rogue Model For Cubic and Bipartite)**.** *Let M be a rogue model at t of a formula φ' with $\varphi' \supseteq R^{C+B}(\varphi, \mathcal{T})$. Assume that (1) there is no ancestor t' of t in T such that M is rogue at t' and that (2) t is on the lexicographic smallest root-to-leaf path of T. The* symmetric rogue model M' (of M) *is constructed as follows.*

- *If $x \notin M$, we define $M' = M$*
- *Otherwise, if $x \in M$:*
  - *Replace $o_t \in M$ by $e_t \in M'$, $o'_t \in M$ by $e'_t \in M'$, and $o''_t \in M$ by $e''_t \in M'$ (as well as vice versa, i.e., $e_t \in M$ iff $o_t \in M'$, $e'_t \in M$ iff $o'_t \in M'$, and $e''_t \in M$ iff $o''_t \in M'$).*
  - *For every ancestor t' of t in T, we replace $o_{t'} \in M$ by $e_{t'} \in M'$, $o'_{t'} \in M$ by $e'_{t'} \in M'$, $o''_{t'} \in M$ by $e''_{t'} \in M'$, and vice versa (i.e., $e_{t'} \in M$ iff $o_{t'} \in M'$, $e'_{t'} \in M$ iff $o'_{t'} \in M'$, $e''_{t'} \in M$ iff*

$o''_{t'} \in M'$), as well as $o1_{t'} \in M$ by $e1_{t'} \in M'$, $e1_{t'} \in M$ by $o1_{t'} \in M'$, $o2_{t'} \in M$ by $e2_{t'} \in M'$, $e2_{t'} \in M$ by $o2_{t'} \in M'$ (and vice versa)

- *If (a) either $o_t \in M$ or $e_t \in M$, and (b) $|M \cap \{o1_t, o2_t, e1_t, e2_t\}| \geq 1$, we additionally replace $o1_t \in M$ by $e2_t \in M'$, $o2_t \in M$ by $e1_t \in M'$ and vice versa (i.e., $e1_t \in M$ by $o2_t \in M'$, $e2_t \in M$ by $o1_t \in M'$).*
- *If (a), (b), and (c) $|M \cap \{\underset{\cdot}{c}, \underset{\cdot}{\overline{c}}\}| = 1$ with $c \in \delta(t)$, we additionally replace $\underset{\cdot}{c} \in M$ by $\underset{\cdot}{\overline{c}} \in M'$, $c' \in M$ by $\overline{c}' \in M'$, $c'' \in M$ by $\overline{c}'' \in M'$ and vice versa (i.e., $\underset{\cdot}{\overline{c}} \in M$ by $\underset{\cdot}{c} \in M'$, $\overline{c}' \in M$ by $c' \in M'$, $\overline{c}'' \in M$ by $c'' \in M'$).*

We say that M is the symmetric rogue model *of M'.*

With these key definitions, we can establish correctness similarly to Lemmas 28–30 and Proposition 31. There, the crucial observation is that we can always perform the translations required by the symmetric rogue model of Definition 42. Indeed, even if there is only one of $\underset{\cdot}{c}$ or $\underset{\cdot}{\overline{c}}$ in a model M, in M' we still need to precisely flip between copy variables for $\underset{\cdot}{c}$ and those for $\underset{\cdot}{\overline{c}}$ (see the added last case in Definition 42).

9. Proofs for New Characterization of GapP

Theorem 7 (Characterization of GapP)**.**

$$\begin{aligned}&\mathrm{gapP}\\&=[\#2\textsc{sat}-\#2\textsc{sat}]^{\log}=[\#\textsc{impl2sat}-\#\textsc{impl2sat}]^{\log}\\&=[\#0{,}1\text{-}2\textsc{dnf}-\#0{,}1\text{-}2\textsc{dnf}]^{\log}=[\#2\textsc{dnf}-\#2\textsc{dnf}]^{\log}\\&=[\#\textsc{dnf}-\#\textsc{dnf}]^{\log}=[\#\textsc{mon2sat}-\#\textsc{mon2sat}]^{\log}\\&=[\#\textsc{mon2dnf}-\#\textsc{mon2dnf}]^{\log}=\mathrm{spanL}-\mathrm{spanL}.\end{aligned}$$

The characterization extends to cubic *and* bipartite *restrictions of* #IMPL2SAT *and* #0,1-2DNF*; even if both formulas use the same variables and differ by only one literal/variable occurrence.*

Proof. Class gapP is equivalent to the subtraction of two #P calls [19, Proposition 3.5]. We show the inclusions from left to right (and then close the cycle).

Case "gapP $\subseteq$ $[\#2\textsc{sat} - \#2\textsc{sat}]^{\log}$": Since gapP is equivalent to #P − #P, it is equivalent to $[\#3\textsc{sat} - \#3\textsc{sat}]^{\log}$. Indeed, each of these #P calls can be parsimoniously translated into #(3)SAT [41, Lemma 3.2] and it is easy to see that these translations can be carried out using a constant number of pointers to the input. This results in two formulas φ and φ'. Then, we apply our reduction as in Theorem 3 on φ, resulting in IMPL2SAT formulas ψ_1, ψ_2 such that $\#(\varphi) = \#(\psi_1) - \#(\psi_2)$. Similarly, we obtain ψ_1' and ψ_2' from φ'. We compute $\#(\varphi) - \#(\varphi')$ by $(\#\psi_1 - \#\psi_2) - (\#\psi_1' - \#\psi_2') = (\#\psi_1 + \#\psi_2') - (\#\psi_2 + \#\psi_1')$. From this, we construct formula $\alpha = \psi_1 \cup \psi_2''$, where ψ_2'' is obtained from ψ_2' by replacing every variable with a fresh variable. Analogously, we construct $\beta = \psi_2 \cup \psi_1''$, where ψ_1'' is obtained from ψ_1' by replacing variables with fresh variables. Observe that $\#(\alpha) = \#(\psi_1) \cdot \#(\psi_2'')$. To go from "$\cdot$" to "$+$", we need to switch between both formulas alternatively. Such a switch between any two sets V_1, V_2 of variables of IMPL2SAT formulas γ_1, γ_2, is modeled by an IMPL2SAT formula $switch(V_1, V_2) = \{(s \to v), (v' \to s) \mid v \in V_1, v' \in V_2\}$, where s is a fresh variable. Observe that $v' \to s$ is equivalent to $\neg s \to \neg v'$ by contraposition, since $(\neg\neg s \vee \neg v') = (\neg v' \vee s) = (v' \to s)$. Since both γ and δ are in IMPL2SAT, depending on s, we set the variables of γ_1 to 1 (if $s = 1$) or we set variables of γ_2 to 0 (if $s = 0$), as in both cases the formulas are satisfied by the default value. Indeed, with $\alpha' = \alpha \cup switch(\mathrm{vars}(\psi_1), \mathrm{vars}(\psi_2''))$ and $\beta' = \beta \cup switch(\mathrm{vars}(\psi_2), \mathrm{vars}(\psi_1''))$ we establish the claim, as $\#(\alpha') - \#(\beta') = \#(\varphi) - \#(\varphi')$.

Case "$[\#2\textsc{sat} - \#2\textsc{sat}]^{\log} \subseteq [\#\textsc{impl2sat} - \#\textsc{impl2sat}]^{\log}$": Without loss of generality, we may assume two formulas ψ and ψ' in IMPL2SAT, where the goal is to compute $\#(\psi) - \#(\psi')$. Indeed, if either ψ or ψ' were not in IMPL2SAT, it can be translated as shown above.

Case "$[\#\textsc{impl2sat} - \#\textsc{impl2sat}]^{\log} \subseteq [\#0{,}1\text{-}2\textsc{dnf} - \#0{,}1\text{-}2\textsc{dnf}]^{\log}$": Assume two formulas ψ and ψ' in IMPL2SAT, where the goal is to compute $\#(\psi) - \#(\psi')$. The goal is to obtain this number by computing $2^{|\mathrm{vars}(\psi)|} - \#(\neg\psi) - (2^{|\mathrm{vars}(\psi')|} - \#(\neg\psi'))$. However, in general, the number of variables of ψ might differ from those in ψ'. To compensate for this difference $n = |\mathrm{vars}(\psi) - \mathrm{vars}(\psi')|$, we need to add additional variables to the smaller formula. Let α, β be the formulas ψ, ψ' such that $|\mathrm{vars}(\alpha)| < |\mathrm{vars}(\beta)|$. We will reuse the $switch$ construction from above, where we let $V = \{v_1, \ldots, v_n\}$ be fresh variables, $\alpha' = \alpha \cup switch(\mathrm{vars}(\alpha) \cup V, V)$ and $\beta' = \beta \cup switch(\mathrm{vars}(\beta), \emptyset)$. Observe that due to $switch$, both α' and β' have one additional satisfying assignment. Indeed, the construction ensures that $2^{|\mathrm{vars}(\alpha')|} - \#(\neg\alpha') - (2^{|\mathrm{vars}(\beta')|} - \#(\neg\beta')) = \#(\neg\beta') - \#(\neg\alpha') = \#(\neg\psi') - 1 - (\#(\neg\psi) - 1) = \#(\neg\psi') - \#(\neg\psi) = \#(\psi) - \#(\psi')$.

Case "$[\#0{,}1\text{-}2\textsc{dnf} - \#0{,}1\text{-}2\textsc{dnf}]^{\log} \subseteq [\#2\textsc{dnf} - \#2\textsc{dnf}]^{\log}$": By definition.

Case "$[\#2\textsc{dnf} - \#2\textsc{dnf}]^{\log} \subseteq [\#3\textsc{dnf} - \#3\textsc{dnf}]^{\log}$": By definition.

Case "$[\#3\textsc{dnf} - \#3\textsc{dnf}]^{\log} \subseteq [\#\textsc{mon2sat} - \#\textsc{mon2sat}]^{\log}$": As mentioned above, there is a parsimonious many-one reduction from any problem in #P to #SAT, as the Cook-Levin construction is solution preserving [41, Lemma 3.2], which works in logspace. Consequently, for gapP we obtain two propositional formulas φ, φ' such that the goal is to compute $c = \#(\varphi) - \#(\varphi')$. Then, we use our reduction R on φ, resulting in two #MON2CNF formulas ψ_1, ψ_2. Similarly, applying R on φ' yields the #MON2CNF formulas ψ_1', ψ_2'. Then, we have that $c = \#(\psi_1) - \#(\psi_2) - (\#(\psi_1') - \#(\psi_2')) = (\#(\psi_1) + \#(\psi_2')) - (\#(\psi_2) + \#(\psi_1'))$. Now, it remains to construct formulas α, α' such that $\#(\alpha) - \#(\alpha') = c = (\#(\psi_1) + \#(\psi_2')) - (\#(\psi_2) + \#(\psi_1'))$. To this end, we build a formula $monswitch(\iota, \tau, \kappa)$ over three formulas ι, τ, κ (such that $\iota \subseteq \kappa$, $\tau \subseteq \kappa$), which uses fresh variables s_ι, s_τ and constructs $s_\iota \vee v$ for every $v \in \mathrm{vars}(\kappa \setminus \iota)$, as well as $s_\tau \vee v'$ for every $v' \in \mathrm{vars}(\kappa \setminus \tau)$. This ensures that if s_ι is set to false, we obtain $\#(\iota)$ models, whereas setting s_τ to false, we receive $\#(\tau)$ models. If both s_ι and s_τ are set to true, the goal is to obtain $\#(\kappa)$ many models, and if both s_ι and s_τ are set to false, we get 1 model.

Without loss of generality, we assume that $\psi_1, \psi_2, \psi_1', \psi_2'$ do not share variables, which can be easily achieved by renaming. Let $\beta = \psi_1 \cup \psi_2 \cup \psi_1' \cup \psi_2'$. Then, we build $\alpha = \beta \cup monswitch(\psi_1, \psi_2', \beta)$ as well as $\alpha' = \beta \cup monswitch(\psi_2, \psi_1', \beta)$. Consequently, we have that $\#(\alpha) = \#(\psi_1) + \#(\psi_2') + \#(\beta) + 1$ and $\#(\alpha') = \#(\psi_2) + \#(\psi_1') + \#(\beta) + 1$, resulting in $c = \#(\alpha) - \#(\alpha')$.

Case "$[\#\text{MON2SAT} - \#\text{MON2SAT}]^{\log} \subseteq [\#\text{MON2DNF} - \#\text{MON2DNF}]^{\log}$": Similar to above we assume two formulas ψ and ψ' in MON2SAT, with the goal of computing $\#(\psi) - \#(\psi')$. Then, this equals to $2^{|\operatorname{vars}(\psi)|} - \#(\neg\psi) - (2^{|\operatorname{vars}(\psi')|} - \#(\neg\psi'))$. However, in general, the number of variables of ψ might differ from those in ψ'. To compensate, adding additional variables to the smaller formula seems challenging (without negation). However, we can do the following. Without loss of generality, assume that ψ and ψ' do not share variables and let $\beta = \psi \cup \psi'$. We will reuse the $monswitch$ construction from above, where we construct $\alpha = \psi \cup monswitch(\psi, \emptyset, \beta)$ and $\alpha' = \psi' \cup monswitch(\psi', \emptyset, \beta)$. Indeed, the construction ensures that $\#(\neg\alpha') - \#(\neg\alpha) = 2^{|\operatorname{vars}(\alpha')|} - \#(\alpha') - (2^{|\operatorname{vars}(\alpha)|} - \#(\alpha)) = \#(\alpha) - \#(\alpha') = \#(\psi) + 1 + 1 + \#(\psi)\#(\psi') - (\#(\psi') + 1 + 1 + \#(\psi)\#(\psi')) = \#(\psi) - \#(\psi')$.

Case "$[\#\text{MON2DNF} - \#\text{MON2DNF}]^{\log} \subseteq \text{spanL} - \text{spanL} \subseteq \text{gapP}$": Trivial, since $\#\text{DNF} \in \text{spanL}$ [2, Proof of Theorem 4.8]. Further, we have $\text{spanL} - \text{spanL} \subseteq \#\text{P} - \#\text{P} = \text{gapP}$ since $\text{spanL} \subseteq \#\text{P}$ and gapP is equivalent to the subtraction of two $\#\text{P}$ calls [19, Proposition 3.5].

Then, we can follow the chain of inclusions above to finally establish the claim.

In order to establish stronger claims involving properties bipartiteness and max-degree 3 for #IMPL2SAT and #0,1-2DNF, we replace the $switch$ construction above. We provide $cycswitch$ over formulas φ_1 and φ_2 that preserves the required properties (at most degree 3 and bipartiteness: the edges of primal graphs G_{φ_1} and G_{φ_2} are in $V_1^e \times V_1^o$ and $V_2^e \times V_2^o$, respectively). The idea is precisely the same as above, but the actual construction, referred to by $cycswitch(V_1, V_2)$ with $V_1 = \operatorname{vars}(\varphi_1)$ and $V_2 = \operatorname{vars}(\varphi_2)$, is more involved. Indeed, to preserve both properties of degree at most 3 and bipartiteness, we will construct chains of implications, using $4m$ fresh switch variables of the form $S = \{s_i^e, s_i^o \mid 1 \leq i \leq 2m\}$ (where $m = \max(|V_1|, |V_2|)$), as well as 5 copy variables $v^1, \ldots, v^5$ for every variable v in $V_1 \cup V_2$ that is of degree 3.

The switch consists of cyclic implications of the form $(s_1^e \to s_1^o), (s_1^o \to s_2^e), \ldots, (s_{2m}^e \to s_{2m}^o), (s_{2m}^o \to s_1^e)$. This ensures that either each of these bits is set to 1 or all are set to 0. Variables $u_i \in V_1$ of degree ≤ 2 can be easily connected to the switch cycle, using implications $(s_i^e \to u_i)$ if $u_i \in V_1^e$, and $(s_i^o \to u_i)$ if $u_i \in V_1^o$. Analogously, variables $u_i \in V_2$ of degree ≤ 2 can be connected using implications $(u_i \to s_{m+i}^e)$ if $u_i \in V_2^e$, and $(u_i \to s_{m+i}^o)$ if $u_i \in V_2^o$. For variables $v_j \in V_1$ of degree 3, we need to rewrite such that we can reduce to the case of degree 2 above. Let therefore w_1 and w_2 be two neighbors of v_j in G_{φ_1} such that both w_1, w_2 form outgoing implications (to v_j) or incoming implications (to w_1/w_2). For the sake of concreteness, we assume both $(w_1 \to v_j)$ and $(w_2 \to v_j)$ are in φ_1, as the other case $(v_j \to w_1)$ and $(v_j \to w_2)$ works analogously (which covers all cases we need to consider). We additionally construct $(v_j \to v_j^1)$, $(v_j^1 \to v_j^2)$, $(v_j^2 \to v_j^3)$, $(v_j^3 \to v_j^4)$, $(v_j^4 \to v_j^5)$, and $(v_j^5 \to v_j)$. Then, we can construct implications $(w_1 \to v_j^2)$ and $(w_2 \to v_j^4)$. As above, observe that v_j^1 is of degree 2 and can be connected to the switch cycle.

In the end, for every such variable v_j we also need to remove implications $(w_1 \to v_j)$ and $(w_2 \to v_j)$ from φ_1, resulting in φ_1'. Analogously, we proceed for variables $v_j \in V_2$, construct implications in $cycswitch(V_1, V_2)$, and remove implications from φ_2 as above, resulting in φ_2'. The overall construction preserves max. degree 3 and bipartiteness. $\square$

10. Proofs for A New Characterization of PH

Theorem 13. $\mathrm{gapP}\subseteq[\#\mathrm{IMPL2SAT}]^{\log}_{\mathrm{AC}^0}=[\#0,1\text{-}2\mathrm{DNF}]^{\log}_{\mathrm{AC}^0}$. *This statement holds even if* #IMPL2SAT *is restricted to cubic and bipartite formulas.*

Proof. "$\mathrm{gapP} \subseteq [\#\mathrm{IMPL2SAT}]^{\log}_{\mathrm{AC}^0}$": In contrast to above, we instead apply Theorems 3 (B) and 7, thereby obtaining two IMPL2CNF formulas φ_1 and φ_2, which have the desired properties. We assume that $\mathrm{vars}(\varphi_1) \cap \mathrm{vars}(\varphi_2) = \emptyset$, which can be obtained by renaming all variables. Let $n = |\mathrm{vars}(\varphi_1)|$ and $n' = |\mathrm{vars}(\varphi_2)|$. We only need to find a basis that is provably larger than $\max(\#(\varphi_1), \#(\varphi_2))$, namely $> 2^{\max(n,n')}$. Therefore, $m = \max(n, n')$ bits are sufficient.

Similarly to the switch construction in Theorem 12 above, we need to provide an updated switch version. We extend $cycswitch$, as defined in the proof of Theorem 7, that preserves the required properties (at most degree 3 and bipartiteness: the edges of primal graphs G_{φ_1} and G_{φ_2} are in $V_1^e \times V_1^o$ and $V_2^e \times V_2^o$, respectively). The idea is precisely the same as above, but the actual construction, referred to by $extcycswitch(B, V_1, V_2)$ with $V_1 = \mathrm{vars}(\varphi_1)$ and $V_2 = \mathrm{vars}(\varphi_2)$, is more involved. As above, we will construct chains of implications, using $4m$ fresh switch variables of the form $S = \{s_i^e, s_i^o \mid 1 \leq i \leq 2m\}$, m bit variables $B = \{b_1, \dots, b_m\}$ as above, as well as 5 copy variables $v^1, \dots, v^5$ for every variable v in $V_1 \cup V_2$ of degree 3.

The switch $extcycswitch$ consists of implications of the form $(s_1^e \rightarrow s_1^o), (s_1^o \rightarrow s_2^e), \dots, (s_{2m}^e \rightarrow s_{2m}^o), (s_{2m}^o \rightarrow s_1^e)$. Variables $u_i \in V_1$ of degree ≤ 2 can be easily connected to the switch cycle, using implications $(s_i^e \rightarrow u_i)$ if $u_i \in V_1^e$, and $(s_i^o \rightarrow u_i)$ if $u_i \in V_1^o$. Analogously, variables $u_i \in V_2$ of degree ≤ 2 can be connected using implications $(u_i \rightarrow s_{m+i}^e)$ if $u_i \in V_2^e$, and $(u_i \rightarrow s_{m+i}^o)$ if $u_i \in V_2^o$. Again, variables $v_j \in V_1$ of degree 3 need to be rewritten such that we can reduce to the case of degree 2 above. Let w_1 and w_2 be two neighbors of v_j in G_{φ_1} such that w_1, w_2 form outgoing implications (to v_j) or incoming implications (to w_1/w_2). For concreteness, assume both $(w_1 \rightarrow v_j)$ and $(w_2 \rightarrow v_j)$ are in φ_1; the other case $(v_j \rightarrow w_1)$ and $(v_j \rightarrow w_2)$ works analogously. We additionally construct $(v_j \rightarrow v_j^1)$, $(v_j^1 \rightarrow v_j^2)$, $(v_j^2 \rightarrow v_j^3)$, $(v_j^3 \rightarrow v_j^4)$, $(v_j^4 \rightarrow v_j^5)$, and $(v_j^5 \rightarrow v_j)$. We construct implications $(w_1 \rightarrow v_j^2)$ and $(w_2 \rightarrow v_j^4)$; now v_j^1 is of degree 2 and can be connected to the switch cycle.

For every variable v_j we need to remove implications $(w_1 \rightarrow v_j)$ and $(w_2 \rightarrow v_j)$ from φ_1, resulting in φ_1'. Analogously, we proceed for variables $v_j \in V_2$, construct implications in $extcycswitch(B, V_1, V_2)$, and remove implications from φ_2 as above, resulting in φ_2'. Finally, we need to connect B to the switch. Since there are at least $3m$ possibilities in total (and connecting the variables in $V_1 \cup V_2$ already used up at most $2m$ of them), this can be achieved by connecting any b_i in B to a variable s_j^* in S that is not yet connected to a variable in $V_1 \cup V_2$.

Consequently, we construct the formula $\alpha = \varphi_1' \cup \varphi_2' \cup extcycswitch(B, \mathrm{vars}(\varphi_1), \mathrm{vars}(\varphi_2))$. Then, if every variable in S is set to 1, $\#(\alpha)$ corresponds to the number of models of φ_2. Otherwise, every variable in S is set to 0 by construction, yielding $2^m\#(\varphi_1)$ many assignments. This results in $\#(\alpha) = \#(\varphi_2)+2^m\cdot\#(\varphi_1)$. It is easy to see that this reduction works in logspace, using a constant number of pointers to the input.

After counting, we can integer-divide the result by 2^m, and obtain the result $\#(\varphi_1)$ as well as the remainder $\#(\varphi_2)$ of the division. As above, this works in AC^0 using bit-wise AND (see also Theorem 14). Finally, the result is established by a single subtraction, computing $\#(\varphi_1) - \#(\varphi_2)$ in AC^0, which establishes the required result.

"$[\#\mathrm{IMPL2SAT}]^{\log}_{\mathrm{AC}^0} = [\#0,1\text{-}2\mathrm{DNF}]^{\log}_{\mathrm{AC}^0}$": Follows from Lemma 27. □

Theorem 12. $\mathrm{gapP}\subseteq[\#\mathrm{MON2SAT}]^{\log}_{\mathrm{TC}^0}=[\#\mathrm{MON2DNF}]^{\log}_{\mathrm{TC}^0}$.

Proof. "$\mathrm{gapP} \subseteq [\#\mathrm{MON2SAT}]^{\log}_{\mathrm{TC}^0}$": We use Theorems 3 (A) and 7 to obtain two MON2CNF formulas φ_1 and φ_2. Then, we rename all the variables in φ_2, obtaining φ_2', which does not share a variable with φ_1. Let $n = |\mathrm{vars}(\varphi_1)|$ and $n' = |\mathrm{vars}(\varphi_2)|$. Now, in order to represent both formulas in a single call, we need to find a basis that is provably larger than the product $\#(\varphi_1) \cdot \#(\varphi_2)$, namely $> 2^{n+n'}$. Therefore, $m = n+n'+1 > n+n'$ bits are sufficient. We will construct a (relaxed) version of the switch construction, as used in the proof of Theorem 7. This construction adds clauses of the form $s\vee b_1, \dots, s\vee b_m$ for fresh variables $s, b_1, \dots, b_m$, thereby ensuring that if s is assigned to 0, all the bits of the basis must be fixed (set to 1). Further, we add $s \vee v_1, \dots, s \vee v_n$ for the variables $\{v_1, \dots, v_n\} = \mathrm{vars}(\varphi_1)$ of the first formula φ_1. This ensures that if s is 0, we only obtain satisfying assignments of φ_2. More precisely, for any two sets B, V of variables, we construct $relswitch(B, V) = \{(s \vee b), (s \vee v) \mid b \in B, v \in V\}$.

Consequently, we construct the formula $\alpha = \varphi_1 \cup \varphi_2' \cup relswitch(\{b_1, \dots, b_m\}, \mathrm{vars}(\varphi_1))$. Then, if $s = 1$, $\#(\alpha)$ corresponds to the number of models of φ_1 multiplied by those of φ_2 multiplied by 2^m. This results in $\#(\alpha) = \#(\varphi_2) + \#(\varphi_1)\cdot\#(\varphi_2)\cdot 2^m$. It is easy to see that this reduction works in logspace, as we only need a constant number of pointers to the input. After counting, we can integer-divide the result by 2^m, and obtain the remainder $\#(\varphi_2)$ of the division, which works in AC^0 using bit-wise AND on $\#(\alpha)$ (see also Theorem 14). Then, we obtain the integer part $\#(\varphi_1) \cdot \#(\varphi_2)$ of the division (also in AC^0 using bit-wise AND). Finally, by dividing the integer part by the remainder $\#(\varphi_2)$ we can reconstruct $\#(\varphi_1) = \frac{\#(\varphi_1)\cdot\#(\varphi_2)}{\#(\varphi_2)}$, which requires TC^0 [25]. Finally, the result is obtained by a single subtraction, computing $\#(\varphi_1) - \#(\varphi_2)$ in AC^0, which establishes the required result.

"$[\#\mathrm{MON2SAT}]^{\log}_{\mathrm{TC}^0} = [\#\mathrm{MON2DNF}]^{\log}_{\mathrm{TC}^0}$": Follows from Lemma 27. □

Theorem 14 (Characterization of PH)**.**

$$\mathrm{PH} \subseteq [\#\mathrm{MON2SAT}]^{\log}_{\mathrm{TC}^0} = [\#\mathrm{MON2DNF}]^{\log}_{\mathrm{TC}^0} \subseteq \mathrm{P}^{\#2\mathrm{DNF}[1]}$$

$$\mathrm{PH} \subseteq [\#\mathrm{IMPL2SAT}]^{\log}_{\mathrm{AC}^0} = [\#0,1\text{-}2\mathrm{DNF}]^{\log}_{\mathrm{AC}^0} \subseteq \mathrm{P}^{\#2\mathrm{DNF}[1]}$$

Proof. We perform the known reduction [39] from PH to $\mathrm{P}^{\#\mathrm{P}[1]}$ that uses a single #SAT call, followed by computing the remainder of a division by 2^m, where m is polynomial in the size of the input. The whole reduction can be computed in logspace. The key ingredient [39] is actually Lemma 2.1 of the Valiant-Vazirani theorem [43]. However, each step does not only work in linear time (as claimed), but there is no need to keep more than a constant number of pointers to the input to output the formula since w is picked randomly for each round.

Then, we apply Theorem 12, obtaining a single #MON2SAT formula φ and encode the 2 shifting operations, 1 division, as well as 1 subtraction into the TC^0 circuit [25] for postprocessing. Finally, we also encode the final division by 2^m into the TC^0 circuit, such that the result equals 1 iff the result does not divide by 2^m, i.e., $\not\equiv 0(\mathrm{mod} 2^m)$. Observe that this operation actually works in AC^0. Indeed, one can encode this via a binary AND operation with a bit-mask where every bit is set to 1, except the m least significant bits, which are set to 0. The inclusion $[\#\mathrm{MON2SAT}]^{\log}_{\mathrm{TC}^0} \subseteq \mathrm{P}^{\mathrm{spanL}}$ is easy to see, as #DNF is contained in spanL and $[\#\mathrm{MON2SAT}]^{\log}_{\mathrm{TC}^0} = [\#\mathrm{MON2DNF}]^{\log}_{\mathrm{TC}^0}$, see Lemma 27.

Alternatively, we may apply Theorem 13 to obtain a single #IMPL2SAT formula φ' that has the claimed properties. Then, we can separate both counts using binary AND with a bitmask similar to above (and one on the negated bitmask), which works in AC^0. The resulting subtraction between both parts can also be carried out in AC^0. Finally, we check whether the resulting number is $\not\equiv 0(\bmod 2^m)$, which also works in AC^0 as mentioned above. Fr the closure under negation see Lemma 27. □