On CNF Conversion for SAT and SMT Enumeration

GABRIELE MASINA*, University of Trento, DISI, Italy
GIUSEPPE SPALLITTA, Rice University, USA
ROBERTO SEBASTIANI, University of Trento, DISI, Italy

Modern SAT and SMT solvers are designed to handle problems expressed in Conjunctive Normal Form (CNF) so that non-CNF problems must be CNF-ized upfront, typically by using variants of either Tseitin or Plaisted and Greenbaum transformations. When passing from plain solving to enumeration, however, the capability of producing partial satisfying assignments that are as small as possible becomes crucial, which raises the question of whether such CNF encodings are also effective for enumeration.

In this paper, we investigate both theoretically and empirically the effectiveness of CNF conversions for SAT and SMT enumeration. On the negative side, we show that: (i) Tseitin transformation prevents the solver from producing short partial assignments, thus seriously affecting the effectiveness of enumeration; (ii) Plaisted and Greenbaum transformation overcomes this problem only in part. On the positive side, we prove theoretically and we show empirically that combining Plaisted and Greenbaum transformation with NNF preprocessing upfront —which is typically not used in solving— can fully overcome the problem and can drastically reduce both the number of partial assignments and the execution time.

JAIR Associate Editor: Kuldeep Meel

JAIR Reference Format:
Gabriele Masina, Giuseppe Spallitta, and Roberto Sebastiani. 2025. On CNF Conversion for SAT and SMT Enumeration. *Journal of Artificial Intelligence Research* 83, Article 11 (July 2025), 44 pages. DOI: 10.1613/jair.1.16870

1 Introduction

State-of-the-art SAT and SMT solvers deal very efficiently with formulas expressed in Conjunctive Normal Form (CNF). In real-world scenarios, however, it is common for problems to be expressed as non-CNF formulas. Hence, these problems are converted into CNF before being processed by the solver. This conversion is generally done by using variants of the Tseitin [78] or the Plaisted and Greenbaum [64] transformations, which generate a linear-size equisatisfiable CNF formula by labeling sub-formulas with fresh Boolean atoms. These transformations can also be employed for SAT and SMT enumeration (AllSAT and AllSMT, respectively), by projecting the truth assignments onto the original atoms only.

When passing from plain *solving* to *enumeration*, however, the capability of enumerating *partial* satisfying assignments that are as small as possible is crucial, because each prevents from enumerating a number of total assignments that is exponential w.r.t. the number of unassigned atoms. This raises the question of whether CNF encodings conceived for solving are also effective for enumeration. To the best of our knowledge, however, no research has yet been published to analyze how the different CNF encodings may affect the effectiveness of the solvers for AllSAT and AllSMT.

*Corresponding Author.

Authors' Contact Information: Gabriele Masina, ORCID: 0000-0001-8842-4913, gabriele.masina@unitn.it, University of Trento, DISI, Trento, Italy; Giuseppe Spallitta, ORCID: 0000-0002-4321-4995, gs81@rice.edu, Rice University, Houston, Texas, USA; Roberto Sebastiani, ORCID: 0000-0002-0989-6101, roberto.sebastiani@unitn.it, University of Trento, DISI, Trento, Italy.

DOI: 10.1613/jair.1.16870

Contributions. In this paper, we investigate, both theoretically and empirically, the effectiveness of different CNF transformations for SAT and SMT enumeration, both in the disjoint and non-disjoint cases. The contribution of this paper is twofold.

First, on the negative side, we show that the commonly-employed CNF transformations for solving are not suitable for enumeration. In particular, we notice that the Tseitin transformation introduces top-level label definitions for sub-formulas with double implications, which need to be satisfied as well, and thus prevent the solver from producing short partial assignments. We also notice that the Plaisted and Greenbaum transformation solves this problem only in part —by labeling sub-formulas only with one-way implications if they occur with single polarity— but it has similar issues to the Tseitin transformation when sub-formulas occur with both polarities.

Second, on the positive side, we prove theoretically and we show empirically that converting the formula into Negation Normal Form (NNF) before applying the Plaisted and Greenbaum transformation can fix the problem and drastically improve the effectiveness of the enumeration process by up to orders of magnitude.

This analysis is confirmed by an experimental evaluation of non-CNF problems originating from both synthetic and real-world-inspired applications. The results confirm the theoretical analysis, showing that the proposed combination of NNF with the Plaisted and Greenbaum CNF allows for a drastic reduction in both the number of partial assignments and the execution time.

Previous works. A preliminary version of this paper was presented at SAT 2023 [52]. In this paper, we present the following novel contributions:

- we formalize the main claim of the paper and formally prove it (Theorem 1 in §4);
- we extend the analysis to non-disjoint AllSAT and disjoint and non-disjoint AllSMT;
- we extend the empirical evaluation to a much broader set of benchmarks, including also non-disjoint AllSAT and disjoint and non-disjoint AllSMT, which confirm the theoretical results. We extend the evaluation by using also the novel AllSAT and AllSMT enumerators TabularAllSAT and TabularAllSMT, obtaining similar results as with MathSAT. Moreover, we extend the timeout of each job-pair from 1200s to 3600s, providing thus a more informative comparison;
- we present a much more detailed related work section.

Organization. The paper is organized as follows. In §2 we introduce the theoretical background necessary to understand the rest of the paper. In §3 we analyze the problem of the classical CNF transformations when used for AllSAT and AllSMT. In §4 we propose one possible solution, whose effectiveness is evaluated on both synthetic and real-world-inspired benchmarks in §5. The related work is presented in §6. We conclude the paper in §7, drawing some final remarks and indicating possible future work.

2 Background

This section introduces the notation and the theoretical background necessary to understand the content of this paper. We assume the reader is familiar with the basic syntax, semantics, and results of propositional and first-order logics. We briefly summarize the main concepts and results of SAT and SMT, and the fundamental ideas behind SAT and SMT enumeration and projected enumeration implemented by modern AllSAT and AllSMT solvers.

2.1 SAT and SAT Modulo Theories

Notation. In the paper, we adopt the following conventions. We refer to Boolean atoms with capital letters, such as A and B, and to first-order atoms (including Boolean atoms), with Greek letters, e.g., α. The symbols $\mathbf{A} \stackrel{\text{def}}{=} \{A_1, \ldots, A_N\}$ and $\mathbf{B} \stackrel{\text{def}}{=} \{B_1, \ldots, B_M\}$ denote disjoint sets of Boolean atoms, and $\boldsymbol{\alpha} \stackrel{\text{def}}{=} \{\alpha_1, \ldots, \alpha_k\}$ denotes a

set of first-order (including Boolean) atoms. Propositional and first-order formulas are referred to with Greek letters φ, ψ. We write $\varphi(\boldsymbol{\alpha})$ to denote that $\boldsymbol{\alpha}$ is the set of atoms occurring in φ. We denote Boolean constants by $\mathbb{B} \stackrel{\text{def}}{=} \{\top, \bot\}$. Total truth assignments are denoted by η, whereas partial truth assignments are denoted by μ, possibly annotated with superscripts.

Propositional Satisfiability. A *propositional* (also *Boolean*) *formula* φ can be defined recursively as follows. The Boolean constants $\top$ and $\bot$ are formulas; a Boolean atom A and its negation $\neg A$ are formulas, also referred to as *literals*; a connection of two formulas by one of the Boolean connectives $\wedge, \vee, \rightarrow, \leftrightarrow$ is a formula. A disjunction ($\vee$) of literals is called a *clause*. *Propositional satisfiability (SAT)* is the problem of deciding the satisfiability of a propositional formula, i.e., if there is a way to assign truth values to the atoms such that the formula evaluates to $\top$. We refer to standard literature (e.g., [8]) for details.

Satisfiability Modulo Theories. As it is standard in most SMT literature, we restrict to quantifier-free first-order formulas. A first-order term is either a variable, a constant symbol, or a combination of terms using function symbols. A first-order atom is a predicate symbol applied to a tuple of terms (Boolean atoms can be viewed as zero-arity predicates). A first-order formula is either a first-order atom, or a connection of two formulas by one of the Boolean connectives. A first-order theory $\mathcal{T}$ is a (possibly infinite) set of first-order formulas, that provides an intended interpretation of constant, function, and predicate symbols. Examples of theories of practical interest are those of equality and uninterpreted functions ($\mathcal{EUF}$), of linear arithmetic over integer ($\mathcal{LIA}$) or real numbers ($\mathcal{LRA}$), and combinations thereof. We refer to formulas and atoms over symbols defined by $\mathcal{T}$ as $\mathcal{T}$-formulas and $\mathcal{T}$-atoms, respectively.

Satisfiability Modulo the Theory $\mathcal{T}$, also SMT($\mathcal{T}$), is the problem of deciding the satisfiability of a first-order formula with respect to some background theory $\mathcal{T}$. A formula φ is *$\mathcal{T}$-satisfiable* if $\varphi \wedge \mathcal{T}$ is satisfiable in the first-order sense. Otherwise, it is *$\mathcal{T}$-unsatisfiable*. We refer to standard literature (e.g., [3]) for details.

Hereafter, unless otherwise indicated, by *formulas* we mean both Boolean and $\mathcal{T}$-formulas, and by *atoms* we mean both Boolean and $\mathcal{T}$-atoms, for a generic theory $\mathcal{T}$. We assume that formulas are internally represented as single-rooted directed acyclic graphs (DAGs) where internal nodes are labeled with Boolean connectives, and leaves are labeled with literals or atoms. (Some authors call them "circuits", e.g. [21, 20].) In this way, multiple occurrences of the same sub-formula in one formula are represented by only one shared sub-DAG. The *size* of a formula is the number of nodes and arcs of its DAG representation.

Total and partial truth assignments. Given a set $\boldsymbol{\alpha}$ of atoms, a *total truth assignment* is a total map $\eta : \boldsymbol{\alpha} \longmapsto \mathbb{B}$. A *partial truth assignment* is a partial map $\mu : \boldsymbol{\alpha} \longmapsto \mathbb{B}$. Notice that a partial truth assignment represents (aka "covers") 2^K total truth assignments extending it, where K is the number of unassigned atoms in μ. We often represent a truth assignment either as a set, s.t. $\mu \stackrel{\text{def}}{=} \{\alpha \mid \mu(\alpha) = \top\} \cup \{\neg\alpha \mid \mu(\alpha) = \bot\}$, or as a conjunction of literals, s.t. $\mu \stackrel{\text{def}}{=} \bigwedge_{\mu(\alpha)=\top} \alpha \wedge \bigwedge_{\mu(\alpha)=\bot} \neg\alpha$. If $\mu_1 \subseteq \mu_2$ [resp. $\mu_1 \subset \mu_2$] we say that μ_1 is a *sub-assignment* [resp. *strict sub-assignment*] of μ_2 and that μ_2 is a *super-assignment* [resp. *strict super-assignment*] of μ_1. We denote with $\varphi|_\mu$ the *residual of φ under μ*, i.e. the formula obtained by substituting in φ each $\alpha_i \in \boldsymbol{\alpha}$ with $\mu(\alpha_i)$, and by recursively applying the standard propagation rules of truth values through Boolean operators.

Given a set $\boldsymbol{\alpha}$ of atoms and a formula $\varphi(\boldsymbol{\alpha})$, we say that a *[partial or total] truth assignment $\mu : \boldsymbol{\alpha} \longmapsto \mathbb{B}$ propositionally satisfies φ*, denoted as $\mu \models_p \varphi$, iff $\varphi|_\mu = \top$. [1]

A partial truth assignment μ is *minimal* for φ iff $\mu \models_p \varphi$ and every strict sub-assignment $\mu' \subset \mu$ is such that $\mu' \not\models_p \varphi$. A Boolean formula is satisfiable iff there exists a truth assignment propositionally satisfying it. A $\mathcal{T}$-formula is $\mathcal{T}$-satisfiable iff there exists a $\mathcal{T}$-satisfiable truth assignment propositionally satisfying it.

[1] The definition of satisfiability by partial assignment may present some ambiguities for non-CNF and existentially-quantified formulas [68, 57, 69]. Here we adopt the above definition because it is the easiest to implement, the most efficient to compute, and it is the one typically used by state-of-the-art SAT solvers. We refer to [68, 57, 69] for details.

Given two disjoint sets $\boldsymbol{\alpha}, \mathbf{B}$ of atoms, and aformula $\psi(\boldsymbol{\alpha} \cup \mathbf{B})$, we say that a *[partial or total] truth assignment* $\mu^{\boldsymbol{\alpha}} : \boldsymbol{\alpha} \longmapsto \mathbb{B}$ *propositionallysatisfies* $\exists \mathbf{B}.\psi$ iff there exists a total truth assignment $\eta^{\mathbf{B}} : \mathbf{B} \longmapsto \mathbb{B}$ such that $\mu^{\boldsymbol{\alpha}} \cup \eta^{\mathbf{B}} : \boldsymbol{\alpha} \cup \mathbf{B} \longmapsto \mathbb{B}$ propositionally satisfies ψ.

Two formulas φ and ψ are *propositionally equivalent*, denoted as $\varphi \equiv_p \psi$, iff every total truth assignment propositionally satisfying φ also propositionally satisfies ψ, and vice versa.

Negation Normal Form. Given a formula φ, a sub-formula occurs with *positive* [resp. *negative*] *polarity* (also *positively* [resp. *negatively*]) if it occurs under an even [resp. odd] number of nested negations. Specifically, φ occurs positively in φ; if $\neg\varphi_1$ occurs positively [resp. negatively] in φ, then φ_1 occurs negatively [resp. positively] in φ; if $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$ occur positively [resp. negatively] in φ, then φ_1 and φ_2 occur positively [resp. negatively] in φ; if $\varphi_1 \rightarrow \varphi_2$ occurs positively [resp. negatively] in φ, then φ_1 occurs negatively [resp. positively] and φ_2 occurs positively [resp. negatively] in φ; if $\varphi_1 \leftrightarrow \varphi_2$ occurs in φ, then φ_1 and φ_2 both occur positively and negatively in φ.

A formula is in *Negation Normal Form (NNF)* iff it is given only by the recursive applications of $\wedge$ and $\vee$ to literals, i.e., iff all its sub-formulas occur positively, except for literals. A formula can be converted into a propositionally-equivalent NNF formula by recursively rewriting implications $(\varphi_1 \rightarrow \varphi_2)$ as $(\neg\varphi_1 \vee \varphi_2)$ and equivalences $(\varphi_1 \leftrightarrow \varphi_2)$ as $(\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2)$, and then by recursively "pushing down" the negations: $\neg(\varphi_1 \vee \varphi_2)$ as $(\neg\varphi_1 \wedge \neg\varphi_2)$, $\neg(\varphi_1 \wedge \varphi_2)$ as $(\neg\varphi_1 \vee \neg\varphi_2)$ and $\neg\neg\varphi_1$ as φ_1. The following fact holds.

Lemma 1. *Let φ be a formula and $\mathsf{NNF}(\varphi)$ be the NNF formula resulting from converting φ into NNF as described above. Then the size of $\mathsf{NNF}(\varphi)$ is linear w.r.t. the size of φ.*

(Although we believe this fact is well-known, we provide a formal proof in §A.1.) Intuitively, we only need at most 2 nodes for each sub-formula φ_i of φ, representing $\mathsf{NNF}(\varphi_i)$ and $\mathsf{NNF}(\neg\varphi_i)$ for positive and negative occurrences of φ_i respectively. These nodes are shared among up to exponentially-many branches generated by expanding the nested iffs. Notice that Lemma 1 holds *because a DAG representation of NNF is used*, so that we need at most two nodes for each sub-formula, one for each polarity. (If instead one represented the NNF formula as a tree, then the NNF representation would blow up exponentially in the number of nested $\leftrightarrow$'s in the original formula, because each sub-formula $(\varphi_i \leftrightarrow \varphi_j)$ is recursively expanded into $(\neg\varphi_i \vee \varphi_j) \wedge (\varphi_i \vee \neg\varphi_j)$.)

We have the following fact, for which we provide a complete proof in §A.2.

Lemma 2. *Consider a formula φ, and a partial truth assignment μ. Then $\varphi|_\mu = v$ iff $\mathsf{NNF}(\varphi)|_\mu = v$, for $v \in \{\top, \bot\}$.*

Notice that Lemma 2 *is not* a direct consequence of the fact that $\mathsf{NNF}(\varphi)$ is equivalence-preserving, because the above notion of satisfiability by partial assignment is such that if $\varphi_1 \equiv_p \varphi_2$, then $\mu \models_p \varphi_1$ *does not* imply that $\mu \models_p \varphi_2$ [68, 57, 69]. Consider, e.g., $\varphi_1 \stackrel{\text{def}}{=} (A \vee B) \wedge (A \vee \neg B)$ and $\varphi_2 \stackrel{\text{def}}{=} (A \wedge B) \vee (A \wedge \neg B)$, and the partial assignment $\mu \stackrel{\text{def}}{=} \{A\}$. Although $\varphi_1 \equiv_p \varphi_2$, we have that $\varphi_1|_\mu = \top$, but $\varphi_2|_\mu = B \vee \neg B$, which, although *logically* equivalent to $\top$, is *syntactically* different from it.

CNF Transformations. A formula is in *Conjunctive Normal Form (CNF)* iff it is a conjunction of clauses. Numerous CNF transformation procedures, commonly referred to as CNF-izations, have been proposed in the literature. We summarize the three most frequently employed techniques.

The *Classic CNF-ization* ($\mathsf{CNF}_{\mathsf{DM}}$) converts a formula into a propositionally-equivalent formula in CNF by applying DeMorgan's rules. First, it converts the formula into NNF. Second, it recursively rewrites sub-formulas $\varphi_1 \vee (\varphi_2 \wedge \varphi_3)$ as $(\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3)$ to distribute $\vee$ over $\wedge$, until the formula is in CNF. The principal limitation of this transformation lies in the possible exponential growth of the resulting formula compared to the original (e.g., when the formula is a disjunction of conjunctions of sub-formulas), making it unsuitable for modern SAT and SMT solvers (see e.g., [65]).

The *Tseitin CNF-ization* ($\mathsf{CNF}_{\mathsf{Ts}}$) [78] avoids this exponential blow-up by labeling each sub-formula φ_i with a fresh Boolean atom B_i, which is used as a placeholder for the sub-formula. Specifically, it consists in applying recursively bottom-up the rewriting rule $\varphi \implies \varphi[\varphi_i|B_i] \wedge \mathsf{CNF}_{\mathsf{DM}}(B_i \leftrightarrow \varphi_i)$ until the resulting formula is in CNF, where $\varphi[\varphi_i|B_i]$ is the formula obtained by substituting in φ every occurrence of φ_i with B_i.

The *Plaisted and Greenbaum CNF-ization* ($\mathsf{CNF}_{\mathsf{PG}}$) [64] is a variant of the $\mathsf{CNF}_{\mathsf{Ts}}$ that exploits the polarity of sub-formulas to reduce the number of clauses of the final formula. If a sub-formula φ_i appears only with positive [resp. negative] polarity, then it can be labeled with a one-way implication as $\mathsf{CNF}_{\mathsf{DM}}(B_i \rightarrow \varphi_i)$ [resp. $\mathsf{CNF}_{\mathsf{DM}}(B_i \leftarrow \varphi_i)$]; if φ_i occurs with both polarities, then it is labeled as $\mathsf{CNF}_{\mathsf{DM}}(B_i \leftrightarrow \varphi_i)$, as with $\mathsf{CNF}_{\mathsf{Ts}}$.

With both $\mathsf{CNF}_{\mathsf{Ts}}$ and $\mathsf{CNF}_{\mathsf{PG}}$, due to the introduction of the label variables, the final formula does not preserve the propositional equivalence with the original formula but only the equisatisfiability. Moreover, they also have a stronger property. If $\varphi(\boldsymbol{\alpha})$ is a non-CNF formula and $\psi(\boldsymbol{\alpha} \cup \mathbf{B})$ is either the $\mathsf{CNF}_{\mathsf{Ts}}$ or the $\mathsf{CNF}_{\mathsf{PG}}$ encoding of φ, where $\mathbf{B}$ are the fresh Boolean atoms introduced by the transformation, then $\varphi(\boldsymbol{\alpha}) \equiv_p \exists\mathbf{B}.\psi(\boldsymbol{\alpha} \cup \mathbf{B})$.

Most of the modern SAT and SMT solvers do not deal directly with non-CNF formulas, rather they convert them into CNF by using either $\mathsf{CNF}_{\mathsf{Ts}}$ or $\mathsf{CNF}_{\mathsf{PG}}$, and then find truth assignments propositionally satisfying $\varphi(\boldsymbol{\alpha})$ by finding truth assignments propositionally satisfying $\exists\mathbf{B}.\psi(\boldsymbol{\alpha} \cup \mathbf{B})$.

2.2 AllSAT, AllSMT, Projected AllSAT and Projected AllSMT

AllSAT is the task of enumerating all the truth assignments propositionally satisfying a Boolean formula. The task can be found in the literature in two versions: *disjoint* AllSAT, where the assignments are required to be pairwise mutually inconsistent, and *non-disjoint* AllSAT, where they are not. A generalization to the SMT($\mathcal{T}$) case is AllSMT($\mathcal{T}$), defined as the task of enumerating all the *$\mathcal{T}$-satisfiable* truth assignments propositionally satisfying a SMT($\mathcal{T}$) formula. Also in this case, both disjoint or non-disjoint AllSMT($\mathcal{T}$) are possible.

In the following, for the sake of compactness, we present definitions and algorithms referring to AllSAT. An extension to AllSMT($\mathcal{T}$) can be obtained straightforwardly, by substituting "$\mathbf{A}$" with "$\boldsymbol{\alpha}$" and "truth assignments" with "$\mathcal{T}$-satisfiable truth assignments".

Given a formula φ, we denote with $\mathcal{TTA}(\varphi) \stackrel{\text{def}}{=} \{\eta_1, \ldots, \eta_j \ldots, \eta_M\}$ the set of all total truth assignments propositionally satisfying φ. We denote with $\mathcal{TA}(\varphi) \stackrel{\text{def}}{=} \{\mu_1, \ldots, \mu_i \ldots, \mu_N\}$ a set of partial truth assignments propositionally satisfying φ s.t.:

(a) every $\eta \in \mathcal{TTA}(\varphi)$ is a super-assignment of some $\mu \in \mathcal{TA}(\varphi)$;

and, only in the *disjoint* case:

(b) every pair $\mu_i, \mu_j \in \mathcal{TA}(\varphi)$ assigns opposite truth value to at least one atom.

Notice that, whereas $\mathcal{TTA}(\varphi)$ is unique, multiple $\mathcal{TA}(\varphi)$s are admissible for the same formula φ, including $\mathcal{TTA}(\varphi)$. AllSAT is the task of enumerating either $\mathcal{TTA}(\varphi)$ or a set $\mathcal{TA}(\varphi)$. Typically, AllSAT solvers aim at enumerating a set $\mathcal{TA}(\varphi)$ which is as small as possible, since every partial truth assignment prevents from enumerating a number of total truth assignments that is exponential w.r.t. the number of unassigned atoms, so that to save large amounts of computational space and time.

The enumeration of a $\mathcal{TA}(\varphi)$ for a non-CNF formula φ is typically implemented by first converting it into CNF, and then by enumerating its satisfying assignments by means of *Projected AllSAT*. Specifically, let $\varphi(\mathbf{A})$ be a non-CNF formula and let $\psi(\mathbf{A} \cup \mathbf{B})$ be the result of applying either $\mathsf{CNF}_{\mathsf{Ts}}$ or $\mathsf{CNF}_{\mathsf{PG}}$ to φ, where $\mathbf{B}$ is the set of Boolean atoms introduced by either transformation. $\mathcal{TA}(\varphi)$ is enumerated via Projected AllSAT as $\mathcal{TA}(\exists\mathbf{B}.\psi)$, i.e. as a set of (partial) truth assignments over $\mathbf{A}$ that can be extended to total truth assignments satisfying ψ over $\mathbf{A} \cup \mathbf{B}$. We refer the reader to the general schema described in [41], which we briefly recap here for completeness of narration.

Algorithm 1 MINIMIZE-ASSIGNMENT($\psi_i, \eta_i, \mathbf{A}$)

Input: CNF formula ψ_i, total truth assignment η_i s.t. $\eta_i \models_p \psi_i$, set of relevant atoms $\mathbf{A}$:
$\psi_i(\mathbf{A} \cup \mathbf{B}) \stackrel{\text{def}}{=} \psi \wedge \bigwedge_{j=1}^{i-1} \neg\mu_j^{\mathbf{A}}$ if disjoint, $\psi_i \stackrel{\text{def}}{=} \psi$ if non-disjoint
$\eta_i = \eta_i^{\mathbf{A}} \cup \eta_i^{\mathbf{B}}$
Output: minimal partial truth assignment $\mu_i^{\mathbf{A}}$ s.t. $\mu_i^{\mathbf{A}} \cup \eta_i^{\mathbf{B}} \models_p \psi_i$

```
μ_i^A ← η_i^A
for ℓ ∈ μ_i^A do
    if ψ_i|_{μ_i^A \ {ℓ} ∪ η_i^B} = ⊤ then
        μ_i^A ← μ_i^A \ {ℓ}
return μ_i^A
```

Let $\psi(\mathbf{A} \cup \mathbf{B})$ be a CNF formula over two disjoint sets $\mathbf{A}, \mathbf{B}$ of atoms, where $\mathbf{A}$ is a set of *relevant atoms* s.t. we want to enumerate a $\mathcal{TA}(\exists\mathbf{B}.\psi)$. For disjoint AllSAT with minimal models, the solver enumerates one-by-one partial truth assignments $\mu_1, \ldots, \mu_i, \ldots \mu_N$ which comply with point (a) above where each $\mu_i \stackrel{\text{def}}{=} \mu_i^{\mathbf{A}} \cup \eta_i^{\mathbf{B}}$ is s.t.:

(1) (*satisfiability*) $\mu_i \models_p \psi$;
(2) (*disjointness*) for each $j < i$, $\mu_i^{\mathbf{A}}, \mu_j^{\mathbf{A}}$ assign opposite truth values to some atom in $\mathbf{A}$;
(3) (*minimality*) $\mu_i^{\mathbf{A}}$ is *minimal*, meaning that no literal can be dropped from it without losing properties 1 and 2.

For the non-disjoint case, property 2 is omitted, and the reference to it in 3 is dropped. If the minimality of models is not required, property 3 is omitted.

An example of a basic projected AllSAT procedure producing minimal models (implemented e.g. in MATH-SAT5 [15]) works as follows. At each step i, it finds a total truth assignment $\eta_i \stackrel{\text{def}}{=} \eta_i^{\mathbf{A}} \cup \eta_i^{\mathbf{B}}$ s.t. $\eta_i \models_p \psi_i$, where $\psi_i \stackrel{\text{def}}{=} \psi \wedge \bigwedge_{j=1}^{i-1} \neg\mu_j^{\mathbf{A}}$, and then invokes a minimization procedure on $\eta_i^{\mathbf{A}}$ to compute a partial truth assignment $\mu_i^{\mathbf{A}}$ satisfying properties 1, 2 and 3. Then, the solver adds the blocking clause $\neg\mu_i^{\mathbf{A}}$ —to ensure property 2 for the disjoint version and to ensure an exhaustive exploration of the solutions space for the non-disjoint version— and it continues the search. This process is iterated until ψ_{N+1} is found to be unsatisfiable for some N, and the set $\{\mu_i^{\mathbf{A}}\}_{i=1}^{N}$ is returned. The minimization procedure consists in iteratively dropping literals one-by-one from $\eta_i^{\mathbf{A}}$, checking if it still satisfies the formula. The outline of this minimization procedure is shown in Algorithm 1. Each minimization step is $O(\#clauses \cdot \#vars)$.

Notice that, since we are in the context of *projected* AllSAT, the minimization algorithm only minimizes the relevant atoms in $\mathbf{A}$, and the truth value of existentially quantified variables in $\mathbf{B}$ is still used to check the satisfiability of the formula by the current partial assignment. Moreover, in the disjoint case, to enforce the pairwise disjointness between the assignments, ψ_i in Algorithm 1 refers to the original formula conjoined with all current blocking clauses $\bigwedge_{j=1}^{i-1} \neg\mu_j^{\mathbf{A}}$, whereas in the non-disjoint case ψ_i refers to the original formula only, allowing for a more effective minimization while renouncing the disjointness property.

The procedure reduces to non-projected AllSAT if $\mathbf{B} = \emptyset$. The same procedure can be easily generalized to disjoint and non-disjoint AllSMT($\mathcal{T}$), with the only difference that only $\mathcal{T}$-satisfiable total truth assignments η_i are considered.

We stress the fact that the above procedure is reported just as an example, and that the work described in this paper is agnostic w.r.t. the AllSAT or AllSMT procedure used, provided that it is able to produce *partial* assignments.

3 The Impact of CNF Transformations on Enumeration

In this section, we present a theoretical analysis of the impact of different CNF-izations on the enumeration of short partial truth assignments. In particular, we focus on $\mathsf{CNF}_{\mathsf{Ts}}$ [78] and $\mathsf{CNF}_{\mathsf{PG}}$ [64]. We point out how CNF-izing AllSAT problems using these transformations can introduce unexpected drawbacks for enumeration. In fact, we show that the resulting encodings can force the enumerator to produce partial assignments that are larger in size and in number than necessary. In our analysis we refer to AllSAT, but it applies to AllSMT as well by restricting to theory-satisfiable truth assignments. Moreover, the analysis applies to both disjoint and non-disjoint enumeration.

3.1 The Impact of Tseitin CNF Transformation

We show that using the $\mathsf{CNF}_{\mathsf{Ts}}$ transformation [78] can be problematic for enumeration. In particular, we point out a fundamental weakness of $\mathsf{CNF}_{\mathsf{Ts}}$:

Fact 1. *If a partial assignment $\mu^{\mathbf{A}}$ satisfies φ, this does not imply that $\mu^{\mathbf{A}}$ satisfies $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{Ts}}(\varphi)$.*

In fact, we recall that $\mathsf{CNF}_{\mathsf{Ts}}$ works by applying recursively the rewriting step (§2.1):

$$\varphi \Longrightarrow \varphi[\varphi_i|B_i] \wedge (B_i \leftrightarrow \varphi_i) \tag{1}$$

and then by recursively CNF-izing the two conjuncts. A *partial* assignment $\mu^{\mathbf{A}}$ may satisfy a non-CNF formula $\varphi(\mathbf{A})$ because it does not need to assign a truth value to the atoms in *all* sub-formulas of φ. (E.g., $\mu^{\mathbf{A}}$ can satisfy $\varphi \stackrel{\text{def}}{=} \varphi_1 \vee \varphi_2$ without assigning values to the atoms in φ_2 if it satisfies φ_1.) Consider (1) s.t. φ_i is some sub-formula of φ whose atoms are not assigned by $\mu^{\mathbf{A}}$. Although $\mu^{\mathbf{A}}$ satisfies φ, $\mu^{\mathbf{A}}$ does not satisfy $\exists B_i.(\varphi[\varphi_i|B_i] \wedge (B_i \leftrightarrow \varphi_i))$. In fact, to satisfy the second conjunct it is necessary to assign some truth value not only to B_i but also to some of the unassigned atoms in φ_i, so that to make φ_i evaluate to the same truth value assigned to B_i.

As a consequence of Fact 1, given $\mu^{\mathbf{A}}$ satisfying φ, in order to produce an assignment $\mu^{\mathbf{A}'}$ satisfying $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{Ts}}(\varphi)$ the enumerator is most often forced to assign other atoms in $\mathbf{A}$, so that $\mu^{\mathbf{A}'} \supset \mu^{\mathbf{A}}$. Given the fact that the amount of total assignments covered by a partial assignment decreases exponentially with its length (see §2.1), the above weakness causes a blow-up in the number of partial assignments needed to cover all models. This may drastically affect the effectiveness and efficiency of the enumeration.

This is illustrated in the following example, where instead of one single short partial assignment the enumerator is forced to enumerate 9 longer ones.

Example 1. *Consider the propositional formula over $\mathbf{A} \stackrel{\text{def}}{=} \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$:*

$$\varphi \stackrel{\text{def}}{=} \overbrace{(A_1 \wedge A_2)}^{B_1} \vee (\overbrace{(\overbrace{(A_3 \vee A_4)}^{B_2} \wedge \overbrace{(A_5 \vee A_6)}^{B_3})}^{B_4} \leftrightarrow A_7)^{B_5}. \tag{2}$$

φ is not in CNF, and thus it must be CNF-ized before starting the enumeration process. If $\mathsf{CNF}_{\mathsf{Ts}}$ is used, then the following CNF formula is obtained:

$$\begin{aligned}
\mathsf{CNF}_{\mathsf{Ts}}(\varphi) \stackrel{def}{=} &(\neg B_1 \vee \ \ A_1) \wedge (\neg B_1 \vee \ \ A_2) \wedge (\ \ B_1 \vee \neg A_1 \vee \neg A_2) \wedge && //(B_1 \leftrightarrow (A_1 \wedge A_2)) && \text{(3a)}\\
&(\ \ B_2 \vee \neg A_3) \wedge (\ \ B_2 \vee \neg A_4) \wedge (\neg B_2 \vee \ \ A_3 \vee \ \ A_4) \wedge && //(B_2 \leftrightarrow (A_3 \vee A_4)) && \text{(3b)}\\
&(\ \ B_3 \vee \neg A_5) \wedge (\ \ B_3 \vee \neg A_6) \wedge (\neg B_3 \vee \ \ A_5 \vee \ \ A_6) \wedge && //(B_3 \leftrightarrow (A_5 \vee A_6)) && \text{(3c)}\\
&(\neg B_4 \vee \ \ B_2) \wedge (\neg B_4 \vee \ \ B_3) \wedge (\ \ B_4 \vee \neg B_2 \vee \neg B_3) \wedge && //(B_4 \leftrightarrow (B_2 \wedge B_3)) && \text{(3d)}\\
&(\neg B_5 \vee \ \ B_4 \vee \neg A_7) \wedge (\neg B_5 \vee \neg B_4 \vee \ \ A_7) \qquad \wedge && //(B_5 \leftrightarrow (B_4 \leftrightarrow A_7)) && \text{(3e)}\\
&(\ \ B_5 \vee \ \ B_4 \vee \ \ A_7) \wedge (\ \ B_5 \vee \neg B_4 \vee \neg A_7) \qquad \wedge && && \\
&(\ \ B_1 \vee \ \ B_5) && && \text{(3f)}
\end{aligned}$$

where the fresh atoms $\mathbf{B} \stackrel{def}{=} \{B_1, B_2, B_3, B_4, B_5\}$ *label sub-formulas as in* (2).

Consider the (minimal) partial truth assignment:

$$\mu^{\mathbf{A}} \stackrel{def}{=} \{\neg A_3, \neg A_4, \neg A_7\}. \tag{4}$$

$\mu^{\mathbf{A}}$ *satisfies* φ (2)*, even though it does not assign a truth value to the sub-formulas* $(A_1 \wedge A_2)$ *and* $(A_5 \vee A_6)$*. Yet,* $\mu^{\mathbf{A}}$ *does not satisfy* $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{Ts}}(\varphi)$*. In fact, there is no total truth assignment* $\eta^{\mathbf{B}}$ *on* $\mathbf{B}$ *such that* $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p \mathsf{CNF}_{\mathsf{Ts}}(\varphi)$*, because* (3a) *and* (3c) *cannot be satisfied by assigning only variables in* $\mathbf{B}$*; rather, it is necessary to further assign at least one atom in* $\{A_1, A_2\}$ *to satisfy* (3a) *and at least one in* $\{A_5, A_6\}$ *to satisfy* (3c)*.*

Suppose an enumerator assigns first the literals in $\mu^{\mathbf{A}}$ (4)*, which force to assign also* $\mu^{\mathbf{B}} \stackrel{def}{=} \{\neg B_2, \neg B_4, B_5\}$ *due to* (3b)*,* (3d)*, and* (3e) *respectively. Since* $\mu^{\mathbf{A}} \cup \mu^{\mathbf{B}}$ *satisfies all clauses except those in* (3a) *and* (3c)*, then the enumerator needs extending* $\mu^{\mathbf{A}} \cup \mu^{\mathbf{B}}$ *by enumerating the partial assignments on the unassigned atoms* $\{A_1, A_2, A_5, A_6, B_1, B_3\}$ *which satisfy* (3a) *and* (3c)*. Regardless of the search strategy adopted, this requires generating no less than* 9 *satisfying partial assignments on* $\{A_1, A_2, A_5, A_6\}$*.* [2] *For instance, in the case of disjoint AllSAT, instead of the single partial assignment* $\mu^{\mathbf{A}}$ (4)*, the solver may return the following list of 9 partial assignments satisfying* $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{Ts}}(\varphi)$ *which extend* $\mu^{\mathbf{A}}$*:*

$$\begin{array}{llllllll}
 & \overbrace{}^{B_1} & & & & \overbrace{}^{B_3} & & \\
\{\neg A_1, & & \neg A_3, & \neg A_4, & \neg A_5, & \neg A_6, & \neg A_7\} & //\{\neg B_1, \neg B_2, \neg B_3, \neg B_4, \ \ B_5\}\\
\{\neg A_1, & & \neg A_3, & \neg A_4, & \ \ A_5, & & \neg A_7\} & //\{\neg B_1, \neg B_2, \ \ B_3, \neg B_4, \ \ B_5\}\\
\{\neg A_1, & & \neg A_3, & \neg A_4, & \neg A_5, & \ \ A_6, & \neg A_7\} & //\{\neg B_1, \neg B_2, \ \ B_3, \neg B_4, \ \ B_5\}\\
\{\ \ A_1, & \neg A_2, & \neg A_3, & \neg A_4, & \neg A_5, & \neg A_6, & \neg A_7\} & //\{\neg B_1, \neg B_2, \neg B_3, \neg B_4, \ \ B_5\}\\
\{\ \ A_1, & \neg A_2, & \neg A_3, & \neg A_4, & \ \ A_5, & & \neg A_7\} & //\{\neg B_1, \neg B_2, \ \ B_3, \neg B_4, \ \ B_5\}\\
\{\ \ A_1, & \neg A_2, & \neg A_3, & \neg A_4, & \neg A_5, & \ \ A_6, & \neg A_7\} & //\{\neg B_1, \neg B_2, \ \ B_3, \neg B_4, \ \ B_5\}\\
\{\ \ A_1, & \ \ A_2, & \neg A_3, & \neg A_4, & \neg A_5, & \neg A_6, & \neg A_7\} & //\{\ \ B_1, \neg B_2, \neg B_3, \neg B_4, \ \ B_5\}\\
\{\ \ A_1, & \ \ A_2, & \neg A_3, & \neg A_4, & \ \ A_5, & & \neg A_7\} & //\{\ \ B_1, \neg B_2, \ \ B_3, \neg B_4, \ \ B_5\}\\
\{\ \ A_1, & \ \ A_2, & \neg A_3, & \neg A_4, & \neg A_5, & \ \ A_6, & \neg A_7\} & //\{\ \ B_1, \neg B_2, \ \ B_3, \neg B_4, \ \ B_5\}
\end{array} \tag{5}$$

In the case of non-disjoint AllSAT, the solver may enumerate a similar set of partial assignments, with $\{\neg A_2\}$ *instead of* $\{A_1, \neg A_2\}$ *and* $\{A_6\}$ *instead of* $\{\neg A_5, A_6\}$*.* ◇

[2] The set of models for (3a) is $\{\{\ \ B_1, \ \ A_1, \ \ A_2\}, \{\neg B_1, \ \ A_1, \neg A_2\}, \{\neg B_1, \neg A_1, \ \ A_2\}, \{\neg B_1, \neg A_1, \neg A_2\}\}$, which can be covered only either by $\{\{\ \ B_1, \ \ A_1, \ \ A_2\}, \{\neg B_1, \neg A_1\}, \{\neg B_1, \ \ A_1, \neg A_2\}\}$ or by $\{\{\ \ B_1, \ \ A_1, \ \ A_2\}, \{\neg B_1, \neg A_2\}, \{\neg B_1, \neg A_1, \ \ A_2\}\}$ in the case of disjoint enumeration, and by $\{\{\ \ B_1, \ \ A_1, \ \ A_2\}, \{\neg B_1, \neg A_1\}, \{\neg B_1, \neg A_2\}\}$ in the case of non-disjoint enumeration. In all cases, we need no less than 3 distinct partial assignments on $\{A_1, A_2\}$. Similar considerations hold for (3c). Thus, we need no less than $3 \times 3 = 9$ partial assignments on $\{A_1, A_2\} \cup \{A_5, A_6\}$.

3.2 The Impact of Plaisted and Greenbaum CNF Transformation

We point out that also the $\mathsf{CNF}_{\mathsf{PG}}$ transformation [64] suffers for the same weakness as $\mathsf{CNF}_{\mathsf{Ts}}$ —that is, Fact 1 holds also for $\mathsf{CNF}_{\mathsf{PG}}$— although its effects are mitigated.

In fact, we recall that $\mathsf{CNF}_{\mathsf{PG}}$ works by applying recursively the rewriting step (§2.1):

$$\varphi \Longrightarrow \varphi[\varphi_i|B_i] \wedge \left\{ \begin{array}{ll} (B_i \rightarrow \varphi_i) & \text{if } \varphi_i \text{ occurs only positively in } \varphi \\ (B_i \leftarrow \varphi_i) & \text{if } \varphi_i \text{ occurs only negatively in } \varphi \\ (B_i \leftrightarrow \varphi_i) & \text{if } \varphi_i \text{ occurs both positively and negatively in } \varphi \end{array} \right\}, \tag{6}$$

and then by recursively CNF-izing the two conjuncts. As with $\mathsf{CNF}_{\mathsf{Ts}}$, consider (6) s.t. φ_i is some sub-formula of φ whose atoms are not assigned by $\mu^{\mathbf{A}}$.

If φ_i occurs both positively and negatively in φ, then (6) reduces to (1) and $\mathsf{CNF}_{\mathsf{PG}}$ behaves like $\mathsf{CNF}_{\mathsf{Ts}}$, so that $\mu^{\mathbf{A}}$ does not satisfy $\exists B_i.(\varphi[\varphi_i|B_i] \wedge (B_i \leftrightarrow \varphi_i))$.

If instead φ_i occurs only positively [resp. negatively] in φ, then it is possible to extend $\mu^{\mathbf{A}}$ by assigning B_i to $\bot$ [resp. $\top$] to satisfy $(B_i \rightarrow \varphi_i)$ [resp. $(B_i \leftarrow \varphi_i)$] without assigning any atom in φ_i. Thus $\mu^{\mathbf{A}}$ satisfies $\exists B_i.(\varphi[\varphi_i|B_i] \wedge (B_i \rightarrow \varphi_i))$ [resp. $\exists B_i.(\varphi[\varphi_i|B_i] \wedge (B_i \leftarrow \varphi_i))$].

As with $\mathsf{CNF}_{\mathsf{Ts}}$, given $\mu^{\mathbf{A}}$ satisfying φ, in order to produce an assignment $\mu^{\mathbf{A}'}$ satisfying $\exists \mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\varphi)$ the enumerator is most often forced to assign other atoms in $\mathbf{A}$, so that $\mu^{\mathbf{A}'} \supset \mu^{\mathbf{A}}$. We notice, however, that the effect of this problem is mitigated by the presence of single-polarity sub-formulas among those left unassigned by $\mu^{\mathbf{A}}$. As an extreme case, if all sub-formulas in φ occur with single polarity, then no further assignment to atoms in $\mathbf{A}$ is needed.

Example 2. *Consider the formula φ (2) as in Example 1. Suppose that φ is converted into CNF using $\mathsf{CNF}_{\mathsf{PG}}$. Then, we have:*

$$\begin{aligned}
\mathsf{CNF}_{\mathsf{PG}}(\varphi) \stackrel{def}{=} & (\neg B_1 \vee \ \ A_1) \wedge (\neg B_1 \vee \ \ A_2) && \wedge \quad //(B_1 \rightarrow (A_1 \wedge A_2)) && \text{(7a)}\\
& (\ \ B_2 \vee \neg A_3) \wedge (\ \ B_2 \vee \neg A_4) \wedge (\neg B_2 \vee \ \ A_3 \vee \ \ A_4) \wedge && //(B_2 \leftrightarrow (A_3 \vee A_4)) && \text{(7b)}\\
& (\ \ B_3 \vee \neg A_5) \wedge (\ \ B_3 \vee \neg A_6) \wedge (\neg B_3 \vee \ \ A_5 \vee \ \ A_6) \wedge && //(B_3 \leftrightarrow (A_5 \vee A_6)) && \text{(7c)}\\
& (\neg B_4 \vee \ \ B_2) \wedge (\neg B_4 \vee \ \ B_3) \wedge (\ \ B_4 \vee \neg B_2 \vee \neg B_3) \wedge && //(B_4 \leftrightarrow (B_2 \wedge B_3)) && \text{(7d)}\\
& (\neg B_5 \vee \ \ B_4 \vee \neg A_7) \wedge (\neg B_5 \vee \neg B_4 \vee \ \ A_7) && \wedge \quad //(B_5 \rightarrow (B_4 \leftrightarrow A_7)) && \text{(7e)}\\
& (\ \ B_1 \vee \ \ B_5). && && \text{(7f)}
\end{aligned}$$

We remark that (7a) and (7e) are shorter than (3a) and (3e) respectively, since the corresponding sub-formulas $(A_1 \wedge A_2)$ and $((\ldots) \leftrightarrow A_7)$ occur only with positive polarity in φ (2), so that only the one-way implication $(B_i \rightarrow \varphi_i)$ is needed.

As in Example 1, consider the partial assignment $\mu^{\mathbf{A}} \stackrel{def}{=} \{\neg A_3, \neg A_4, \neg A_7\}$ (4) which satisfies φ (2). As before, $\mu^{\mathbf{A}}$ does not satisfy $\exists \mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\varphi)$. In fact, there is no total truth assignment $\eta^{\mathbf{B}}$ on $\mathbf{B}$ such that $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p \mathsf{CNF}_{\mathsf{PG}}(\varphi)$, because (7c) cannot be satisfied by assigning only variables in $\mathbf{B}$; rather, it is necessary to further assign at least one atom in $\{A_5, A_6\}$ to satisfy (7c). Notice that, unlike with Example 1, in order to satisfy (7a) it is sufficient to set $B_1 = \bot$ with no need to assign any atom in $\{A_1, A_2\}$.

Suppose an enumerator assigns first the literals in $\mu^{\mathbf{A}}$, which force it to assign also $\mu^{\mathbf{B}} \stackrel{def}{=} \{\neg B_2, \neg B_4\}$ due to (7b) and (7d) respectively. Since $\mu^{\mathbf{A}} \cup \mu^{\mathbf{B}}$ satisfies all clauses except those in (7a), (7c), and (7f), the enumerator needs extending $\mu^{\mathbf{A}} \cup \mu^{\mathbf{B}}$ by enumerating partial assignments on the unassigned atoms $\{A_1, A_2, A_5, A_6, B_1, B_3, B_5\}$ which satisfy them. Regardless of the search strategy adopted, to satisfy (7c) it is necessary to generate no less than 3 partial assignments on $\{A_5, A_6\}$ (see Example 1); to satisfy (7a), and (7f), instead, the enumerator needs only assigning $B_1 = \bot$, which forces it to assign $B_5 = \top$ due to (7f).

For instance, in the case of disjoint AllSAT, instead of the single partial assignment $\mu^{\mathbf{A}}$, the solver may return the following list of 3 partial assignments satisfying $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\varphi)$ which extend $\mu^{\mathbf{A}}$:

$$\begin{array}{lccccccl}
 & & & \overbrace{}^{B_3} & & & & \\
\{ & \neg A_3, & \neg A_4, & \neg A_5, & \neg A_6, & \neg A_7\} & // \{\neg B_1, \neg B_2, \neg B_3, \neg B_4, & B_5\} \\
\{ & \neg A_3, & \neg A_4, & A_5, & & \neg A_7\} & // \{\neg B_1, \neg B_2, & B_3, \neg B_4, \quad B_5\} \\
\{ & \neg A_3, & \neg A_4, & \neg A_5, & A_6, & \neg A_7\} & // \{\neg B_1, \neg B_2, & B_3, \neg B_4, \quad B_5\}
\end{array} \tag{8}$$

In the case of non-disjoint AllSAT, the solver may enumerate a similar set of partial assignments, with $\{A_6\}$ instead of $\{\neg A_5, A_6\}$ in the third assignment. ◇

Notice that, to maximize the benefits of $\mathsf{CNF}_{\mathsf{PG}}$, the sub-formulas occurring with positive [resp. negative] polarity only must have their label assigned to false [resp. true]. In practice, this can be achieved in part by instructing the solver to split on negative values in decision branches. [3] Even though the solver is not guaranteed to always assign to false these labels, we empirically verified that this heuristic provides a good approximation of this behavior.

4 Enhancing Enumeration via NNF Preprocessing

In this section, we propose a solution to address the shortcomings of $\mathsf{CNF}_{\mathsf{Ts}}$ and $\mathsf{CNF}_{\mathsf{PG}}$ for SAT and SMT enumeration described in §3. The idea is simple: *we transform first the input formula φ into NNF, and then we apply* $\mathsf{CNF}_{\mathsf{PG}}$ *to* $\mathsf{NNF}(\varphi)$. In fact, NNF guarantees that every non-atomic sub-formula of $\mathsf{NNF}(\varphi)$ occurs only positively, because every sub-formula φ_i occurring with double polarity in φ is converted into two syntactically-different sub-formulas $\varphi_i^+ \stackrel{\text{def}}{=} \mathsf{NNF}(\varphi_i)$ and $\varphi_i^- \stackrel{\text{def}}{=} \mathsf{NNF}(\neg\varphi_i)$, each occurring only positively. Thus, when computing $\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$, φ_i^+ and φ_i^- are labelled with two distinct atoms B_i^+ and B_i^- respectively, adding the one-way implications $(B_i^+ \rightarrow \varphi_i^+)$ and $(B_i^- \rightarrow \varphi_i^-)$. (If φ_i occurs only positively [resp. negatively] in φ, then only φ_i^+ [resp. φ_i^-] occurs in $\mathsf{NNF}(\varphi)$, so that only B_i^+ [resp. B_i^-] is introduced and only $(B_i^+ \rightarrow \varphi_i^+)$ [resp. $(B_i^- \rightarrow \varphi_i^-)$] is added.)

We remark that with this preprocessing we maintain the correctness and completeness of the enumeration process, because φ is equivalent to $\mathsf{NNF}(\varphi)$, which is equivalent to $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$. We remark also that we produce a linear-size CNF encoding, since $\mathsf{NNF}(\varphi)$ has linear size w.r.t. φ and $\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$ has linear size w.r.t. $\mathsf{NNF}(\varphi)$ (see §2.1).

We prove the following fact: *if a partial truth assignment $\mu^{\mathbf{A}}$ satisfies φ, then it satisfies also $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$.* (The vice versa holds trivially.) We remark that, as shown in §3, this fact does not hold for $\mathsf{CNF}_{\mathsf{Ts}}$ and $\mathsf{CNF}_{\mathsf{PG}}$.

Theorem 1. *Consider a formula φ, and a* partial *truth assignment $\mu^{\mathbf{A}}$ such that $\mu^{\mathbf{A}} \models_p \varphi$. Then $\mu^{\mathbf{A}} \models_p \exists\mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$.*

Proof. We show that, for every $\mu^{\mathbf{A}}$ such that $\mu^{\mathbf{A}} \models_p \varphi$, there exists a total truth assignment $\eta^{\mathbf{B}}$ such that $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p \mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$. We first show how such a $\eta^{\mathbf{B}}$ can be built, then we prove that it satisfies $\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$ if conjoined with $\mu^{\mathbf{A}}$. In the following, the symbol "$*$" denotes any formula that is not in $\{\top, \bot\}$, so that "$\varphi_i|_{\mu^{\mathbf{A}}} = *$" means that the residual of φ_i under $\mu^{\mathbf{A}}$ is not a constant.

For each sub-formula φ_i of φ, whose positive and negative occurrences in $\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$ are associated with the variables B_i^+ and B_i^- respectively, do:

(a) if $\varphi_i|_{\mu^{\mathbf{A}}} = \top$, and hence $\mathsf{NNF}(\varphi_i)|_{\mu^{\mathbf{A}}} = \top$ and $\mathsf{NNF}(\neg\varphi_i)|_{\mu^{\mathbf{A}}} = \bot$ by Lemma 2, then add $\{B_i^+, \neg B_i^-\}$ to $\eta^{\mathbf{B}}$;
(b) if $\varphi_i|_{\mu^{\mathbf{A}}} = \bot$, and hence $\mathsf{NNF}(\varphi_i)|_{\mu^{\mathbf{A}}} = \bot$ and $\mathsf{NNF}(\neg\varphi_i)|_{\mu^{\mathbf{A}}} = \top$ by Lemma 2, then add $\{\neg B_i^+, B_i^-\}$ to $\eta^{\mathbf{B}}$;
(c) otherwise if $\varphi_i|_{\mu^{\mathbf{A}}} = *$, then add $\{\neg B_i^+, \neg B_i^-\}$ to $\eta^{\mathbf{B}}$;

[3] To exploit this heuristic also for sub-formulas occurring only negatively, the latter can be labeled with a negative label $\neg B_i$ as $(\neg B_i \leftarrow \varphi_i)$.

(If φ_i occurs only positively or negatively, then only assign B_i^+ or B_i^- respectively.)

We prove that $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p \mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$ by induction on the structure of the formula $\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$. Consider a sub-formula φ_i of φ:

(i) if φ_i occurs positively in φ, then

$$\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi)) \stackrel{\text{def}}{=} \mathsf{CNF}_{\mathsf{PG}}(\overbrace{\mathsf{NNF}(\varphi)[\mathsf{NNF}(\varphi_i)|B_i^+]}^{(9a)} \wedge \overbrace{(B_i^+ \rightarrow \mathsf{NNF}(\varphi_i))}^{(9b)}) \tag{9}$$

(ii) if φ_i occurs negatively in φ, then

$$\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi)) \stackrel{\text{def}}{=} \mathsf{CNF}_{\mathsf{PG}}(\overbrace{\mathsf{NNF}(\varphi)[\mathsf{NNF}(\neg\varphi_i)|B_i^-]}^{(10a)} \wedge \overbrace{(B_i^- \rightarrow \mathsf{NNF}(\neg\varphi_i))}^{(10b)}) \tag{10}$$

For each pair of cases we have:

(a)(i) $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p$ (9a) since we substitute $\mathsf{NNF}(\varphi_i)$ with B_i^+ and $\mathsf{NNF}(\varphi_i)|_{\mu^{\mathbf{A}}} = B_i^+|_{\eta^{\mathbf{B}}} = \top$;
$\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p$ (9b) since $\mu^{\mathbf{A}} \models_p \mathsf{NNF}(\varphi_i)$;

(ii) $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p$ (10a) since we substitute $\mathsf{NNF}(\neg\varphi_i)$ with B_i^- and $\mathsf{NNF}(\neg\varphi_i)|_{\mu^{\mathbf{A}}} = B_i^-|_{\eta^{\mathbf{B}}} = \bot$;
$\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p$ (10b) since $\eta^{\mathbf{B}} \models_p \neg B_i^-$;

(b)(i) $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p$ (9a) since we substitute $\mathsf{NNF}(\varphi_i)$ with B_i^+ and $\mathsf{NNF}(\varphi_i)|_{\mu^{\mathbf{A}}} = B_i^+|_{\eta^{\mathbf{B}}} = \bot$;
$\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p$ (9b) since $\eta^{\mathbf{B}} \models_p \neg B_i^+$;

(ii) $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p$ (10a) since we substitute $\mathsf{NNF}(\neg\varphi_i)$ with B_i^- and $\mathsf{NNF}(\neg\varphi_i)|_{\mu^{\mathbf{A}}} = B_i^-|_{\eta^{\mathbf{B}}} = \top$;
$\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p$ (10b) since $\mu^{\mathbf{A}} \models_p \mathsf{NNF}(\neg\varphi_i)$;

(c)(i) $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p$ (9a) since we substitute $\mathsf{NNF}(\varphi_i)$ with B_i^+, and substituting $\mathsf{NNF}(\varphi_i)|_{\mu^{\mathbf{A}}} = *$ with $B_i^+|_{\eta^{\mathbf{B}}} = \bot$ preserves the satisfiability; $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p$ (9b) since $\eta^{\mathbf{B}} \models_p \neg B_i^+$;

(ii) $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p$ (10a) since we substitute $\mathsf{NNF}(\neg\varphi_i)$ with B_i^-, and substituting $\mathsf{NNF}(\neg\varphi_i)|_{\mu^{\mathbf{A}}} = *$ with $B_i^-|_{\eta^{\mathbf{B}}} = \bot$ preserves the satisfiability; $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p$ (10b) since $\eta^{\mathbf{B}} \models_p \neg B_i^-$;

Therefore, $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p \mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$. □

As a consequence of Theorem 1, given $\mu^{\mathbf{A}}$ satisfying φ, the enumerator is no more forced to assign any more atom in $\mathbf{A}$ to satisfy $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$. This prevents the enumerator from producing multiple assignments extending $\mu^{\mathbf{A}}$, avoiding thus the blow-up in the number of assignments for $\mathsf{CNF}_{\mathsf{Ts}}(\varphi)$ and $\mathsf{CNF}_{\mathsf{PG}}(\varphi)$ described in §3. We stress the fact that Theorem 1 is agnostic of the AllSAT (or AllSMT) algorithm adopted, and that it holds for both disjoint and non-disjoint enumeration.

Remark 1. *Unlike with AllSAT or AllSMT, the pre-conversion into NNF is typically never used in plain SAT or SMT* solving, *because it causes the unnecessary duplication of labels B_i^+ and B_i^-, with extra overhead and no benefit for the solver.*

We notice that the proof of Theorem 1 is *constructive*, that is, it not only says that an assignment $\eta^{\mathbf{B}}$ s.t. $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p \mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$ exists, but also it shows how to construct it. In particular, points (a), (b), and (c) implicitly suggest a strategy for assigning the right values to the $\mathbf{B}$ atoms given $\mu^{\mathbf{A}}$, which is based on the iterative applications of the following steps, interleaved with the assignment of values which are forced by residual constraints:

(a) if B_i^+ occurs negatively in already-satisfied clauses only, then add $\{B_i^+, \neg B_i^-\}$ to $\eta^{\mathbf{B}}$;
(b) if B_i^- occurs negatively in already-satisfied clauses only, then add $\{B_i^-, \neg B_i^+\}$ to $\eta^{\mathbf{B}}$;
(c) if B_i^+ [resp. B_i^-] occurs negatively in a not-yet-satisfied clause whose other unassigned literals are all $\mathbf{A}$-literals, then add $\{\neg B_i^+\}$ [resp. $\{\neg B_i^-\}$] to $\eta^{\mathbf{B}}$.

This strategy mimics the application of points (a), (b), and (c) in the proof to the sub-formulas φ_i^+ and φ_i^- of $\mathsf{NNF}(\varphi)$ in a bottom-up fashion, from the leaves to the root. [4]

We also notice that, due to the constraints $(B_i^+ \rightarrow \varphi_i^+)$ and $(B_i^- \rightarrow \varphi_i^-)$ and to the fact that φ_i^+ and φ_i^- are mutually inconsistent by construction, no assignment satisfying $\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$ may assign both B_i^+ and B_i^- to $\top$. Thus, to further improve the efficiency of the enumeration procedure without affecting its outcome, we also add to $\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$ the mutual-exclusion clauses $(\neg B_i^+ \vee \neg B_i^-)$ when both B_i^+ and B_i^- are introduced, which prevent the solver from assigning both B_i^+ and B_i^- to $\top$, and thus from wasting time in exploring inconsistent search sub-trees for residual formulas like $(\ldots \wedge \varphi_i^+ \wedge \varphi_i^- \wedge \ldots)$.

We illustrate the benefit of our proposed technique with the following example.

Example 3. *Consider the formula φ of Example 1. By converting it into NNF, we obtain:*

$$\mathsf{NNF}(\varphi) \stackrel{def}{=} \overbrace{(A_1 \wedge A_2)}^{B_1^+} \vee ((((\overbrace{\neg A_3 \wedge \neg A_4}^{B_2^-}) \vee (\overbrace{\neg A_5 \wedge \neg A_6}^{B_3^-})) \vee A_7)^{B_4^-})^{B_5^+} \wedge (((\overbrace{A_3 \vee A_4}^{B_2^+}) \wedge (\overbrace{A_5 \vee A_6}^{B_3^+}))^{B_4^+} \vee \neg A_7))^{B_6^+})^{B_7^+}$$

Applying $\mathsf{CNF}_{\mathsf{PG}}$ and adding the mutual-exclusion clauses we obtain the CNF formula:

$$\begin{aligned}
\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi)) \stackrel{def}{=} & (\neg B_1^+ \vee \ A_1) \wedge (\neg B_1^+ \vee \ A_2) \wedge && //(B_1^+ \rightarrow (\ A_1 \wedge \ A_2)) && \text{(11a)}\\
& (\neg B_2^- \vee \neg A_3) \wedge (\neg B_2^- \vee \neg A_4) \wedge && //(B_2^- \rightarrow (\neg A_3 \wedge \neg A_4)) && \text{(11b)}\\
& (\neg B_3^- \vee \neg A_5) \wedge (\neg B_3^- \vee \neg A_6) \wedge && //(B_3^- \rightarrow (\neg A_5 \wedge \neg A_6)) && \text{(11c)}\\
& (\neg B_4^- \vee \ B_2^- \vee \ B_3^-) \wedge && //(B_4^- \rightarrow (\ B_2^- \vee \ B_3^-)) && \text{(11d)}\\
& (\neg B_5^+ \vee \ B_4^- \vee \ A_7) \wedge && //(B_5^+ \rightarrow (\ B_4^- \vee \ A_7)) && \text{(11e)}\\
& (\neg B_2^+ \vee \ A_3 \vee \ A_4) \wedge && //(B_2^+ \rightarrow (\ A_3 \vee \ A_4)) && \text{(11f)}\\
& (\neg B_3^+ \vee \ A_5 \vee \ A_6) \wedge && //(B_3^+ \rightarrow (\ A_5 \vee \ A_6)) && \text{(11g)}\\
& (\neg B_4^+ \vee \ B_2^+) \wedge (\neg B_4^+ \vee \ B_3^+) \wedge && //(B_4^+ \rightarrow (\ B_2^+ \wedge \ B_3^+)) && \text{(11h)}\\
& (\neg B_6^+ \vee \ B_4^+ \vee \neg A_7) \wedge && //(B_6^+ \rightarrow (\ B_4^+ \vee \neg A_7)) && \text{(11i)}\\
& (\neg B_7^+ \vee \ B_5^+) \wedge (\neg B_7^+ \vee \ B_6^+) \wedge && //(B_7^+ \rightarrow (\ B_5^+ \wedge \ B_6^+)) && \text{(11j)}\\
& (\ B_1^+ \vee \ B_7^+) \wedge && && \text{(11k)}\\
& (\neg B_2^+ \vee \neg B_2^-) \wedge && //\textit{mutual exclusion for } B_2^+, B_2^- && \text{(11l)}\\
& (\neg B_3^+ \vee \neg B_3^-) \wedge && //\textit{mutual exclusion for } B_3^+, B_3^- && \text{(11m)}\\
& (\neg B_4^+ \vee \neg B_4^-) && //\textit{mutual exclusion for } B_4^+, B_4^- && \text{(11n)}
\end{aligned}$$

Notice that the labels B_1^-, B_5^-, B_6^-, B_7^- and their respective one-way constraints are not introduced, because there is no negative occurrence of the respective sub-formulas in φ.

[4] In fact, we notice that B_i^+ [resp. B_i^-] occurs negatively only in the clauses encoding the $(B_i^+ \rightarrow \varphi_i^+)$ [resp. $(B_i^- \rightarrow \varphi_i^-)$] constraints, because by construction it always occurs positively elsewhere, since it substitutes some sub-formula φ_i^+ [resp. φ_i^-] which occurs only positively in $\mathsf{NNF}(\varphi)$. Also, by construction, we can have at most one negative **B**-literal per clause.

As in Examples 1 and 2, consider the partial assignment $\mu^{\mathbf{A}} \stackrel{def}{=} \{\neg A_3, \neg A_4, \neg A_7\}$ (4) *which satisfies* φ. *First, it is easy to see that* $\mu^{\mathbf{A}}$ *satisfies also* $\mathsf{NNF}(\varphi)$, *in compliance with Lemma 2. Also,* $\mu^{\mathbf{A}}$ *satisfies* $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$, *because* $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p \mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$ *where* $\eta^{\mathbf{B}} \stackrel{def}{=} \{\neg B_1^+, \neg B_2^+, B_2^-, \neg B_3^+, \neg B_3^-, \neg B_4^+, B_4^-, B_5^+, B_6^+, B_7^+\}$.

The above assignment can be produced by adopting the strategy described above. First, we assign the literals in $\mu^{\mathbf{A}}$, *which force to add also* $\{\neg B_2^+, \neg B_4^+\}$ *due to* (11f) *and* (11h) *respectively. Then we add* $\{B_6^+\}$ *by step (a) on* (11i), $\{B_2^-\}$ *by step (b) on* (11b), *and* $\{\neg B_1^+, \neg B_3^-, \neg B_3^+\}$ *by step (c) on* (11a), (11c), (11g) *respectively. These force to add* $\{B_7^+\}$ *and* $\{B_5^+\}$ *due to* (11k) *and* (11j) *respectively. Finally, we add* $\{B_4^-\}$ *by step (b) on* (11d). *Overall, this corresponds to enumerate only the partial assignment* $\mu^{\mathbf{A}}$ *satisfying* $\exists\mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$, *with both disjoint and non-disjoint enumeration:*

$$\{\neg A_3, \neg A_4, \neg A_5\} \quad //\{\neg B_1^+, \neg B_2^+, B_2^-, \neg B_3^+, \neg B_3^-, \neg B_4^+, B_4^-, B_5^+, B_6^+, B_7^+\}. \qquad \diamond$$

Notice that, the shorter is $\mu^{\mathbf{A}}$ w.r.t. a total assignment, the higher is the chance that $\mathsf{CNF}_{\mathsf{Ts}}$ and $\mathsf{CNF}_{\mathsf{PG}}$ force the production of a high number of extra assignments, the more beneficial is the usage of NNF+$\mathsf{CNF}_{\mathsf{PG}}$ which avoids it. This said, if the enumerator is able to produce short partial assignments $\mu^{\mathbf{A}}$ satisfying the formula, we expect a high benefit from using NNF+$\mathsf{CNF}_{\mathsf{PG}}$ instead of $\mathsf{CNF}_{\mathsf{Ts}}$ and $\mathsf{CNF}_{\mathsf{PG}}$; vice versa, if the enumerator produces only total or nearly-total assignments, we expect no or very-limited benefit respectively.

Implementation-wise, the strategy to assign the values of $\eta^{\mathbf{B}}$ described above is difficult to implement inside the current enumerators. Therefore, there is no formal guarantee that a generic enumeration procedure always finds exactly the $\eta^{\mathbf{B}}$ which prevents the generation of longer assignments. For example, the enumeration procedure of §2.2 finds a total truth assignment $\eta^{\mathbf{A}} \cup \eta^{\mathbf{B}}$ that satisfies the formula, and then finds $\mu^{\mathbf{A}} \subseteq \eta^{\mathbf{A}}$ that is minimal w.r.t. that specific $\eta^{\mathbf{B}}$ such that $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models_p \mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$, so that the $\eta^{\mathbf{B}}$ found is not guaranteed to be the one that allows for the most effective minimization of $\mu^{\mathbf{A}}$. Ad-hoc enumeration heuristics could be adopted. Nevertheless, in §5 we show that a very simple heuristic —i.e., force the assignments of false values first to decision atoms— guarantees dramatic improvements w.r.t. previous approaches using two state-of-the-art AllSAT/AllSMT enumerators.

5 Experimental Evaluation

In this section, we experimentally evaluate the impact of different CNF-izations on the disjoint and non-disjoint AllSAT and AllSMT tasks. In order to compare them on fair ground, we have implemented a base version of each from scratch in PySMT [22], avoiding specific optimizations done by the solvers.

We have conducted a very extensive collection of experiments on AllSAT and AllSMT. We tested the encodings on MathSAT, which supports disjoint and non-disjoint AllSAT and AllSMT, and on TabularAllSAT and TabularAllSMT, which support disjoint AllSAT and AllSMT, respectively. The choice of the solvers is motivated in §5.1; the benchmarks are described in §5.2, the information for the reproducibility of the experiments is given in §5.3, and the results are presented in §5.4; finally, in §5.5 we analyze and discuss the application of the encodings to related fields.

5.1 An Analysis of Available Solvers

In order to evaluate the different CNF encodings, we need an AllSAT/AllSMT solver that (i) is publicly available, (ii) takes as input a CNF formula, (iii) allows performing *projected* enumeration, (iv) allows enumerating (disjoint or non-disjoint) *partial* truth assignments; (v) also, the ability of producing *minimal* assignments is valuable although not necessary.

In the literature, we found the following candidate solvers: for AllSAT, we found RELSAT [4], Grumberg [26], SOLALL [42], Jin [33, 34], clasp [24], PicoSAT [7], Yu [79], BC, NBC, and BDD [76], depbdd [75], Dualiza [56], BASolver [80], AllSATCC [43], HALL [21, 20], TabularAllSAT [72, 73], and model-graph [37]. For AllSMT, the only candidates are MathSAT [15], aZ3 [63], and TabularAllSMT [73].

In Appendix B Table 1 we report an analysis of the above features (i)-(v) for each of the above-mentioned solvers. Overall, we observe that only four solvers match all our needs: MathSAT, TabularAllSAT, TabularAllSMT, and Dualiza. MathSAT supports disjoint and non-disjoint AllSAT and AllSMT enumeration of minimal partial assignments, using the blocking-clause-based enumeration strategy of [41] described in §2.2. TabularAllSAT supports disjoint AllSAT enumeration of non-minimal partial assignments, based on chronological backtracking without blocking clauses [72, 73]. TabularAllSMT, its counterpart for AllSMT, supports disjoint AllSMT. Dualiza supports disjoint and non-disjoint AllSAT enumeration of non-minimal partial assignments, based on dual reasoning. Since Dualiza has been shown empirically to perform poorly [20], we decided to leave it out of our experiments, and to focus on MathSAT, TabularAllSAT, and TabularAllSMT only.

5.2 Description of the Problem Sets

We consider five sets of benchmarks of non-CNF formulas coming from different sources, both synthetic and real-world. For AllSAT, we evaluate the different CNF encodings on three sets of benchmarks. The first one consists of synthetic Boolean formulas, which were randomly generated by nesting Boolean operators up to a fixed depth. The second one consists of formulas encoding properties of ISCAS'85 circuits [12, 27] as done in [74]. The third one is a set of benchmarks on combinatorial circuits encoded as And-Inverter Graphs (AIGs) used in [21].

For AllSMT, we consider two sets of benchmarks. The first one consists of synthetic SMT($\mathcal{LRA}$) formulas, which were randomly generated by nesting Boolean and SMT($\mathcal{LRA}$) operators up to a fixed depth. The second one consists of formulas encoding Weighted Model Integration (WMI) problems [59, 60, 70, 71].

With respect to the conference version of the paper [52], we have extended the set of Boolean synthetic benchmarks and added the AIG benchmarks. Moreover, we have added the SMT($\mathcal{LRA}$) benchmarks, and modified the WMI benchmarks so that they contain also $\mathcal{LRA}$ atoms.

The Boolean synthetic benchmarks. The Boolean synthetic benchmarks were generated by nesting Boolean operators $\wedge, \vee, \leftrightarrow$ until some fixed depth d. Internal and leaf nodes are negated with 0.5 probability. Operators in internal nodes are chosen randomly, giving less probability to the $\leftrightarrow$ operator. In particular, $\leftrightarrow$ is chosen with 0.1 probability, whereas the other two are chosen with an equal probability of 0.45. We generated 100 instances over a set of 20 Boolean atoms and depth $d = 8$, 100 instances over a set of 25 Boolean atoms and depth $d = 8$, and 100 instances over a set of 30 Boolean atoms and depth $d = 6$, for a total of 300 instances.

The ISCAS'85 benchmarks. The ISCAS'85 benchmarks are a set of 10 combinatorial circuits used in test generation, timing analysis, and technology mapping [12]. They have well-defined, high-level structures and functions based on multiplexers, ALUs, decoders, and other common building blocks [27]. We generated random instances as described in [74]. In particular, for each circuit, we constrained 60%, 70%, 80%, 90%, and 100% of the outputs to either 0 or 1, for a total of 250 instances.

The AIG benchmarks. The AIG benchmarks are a set of formulas encoded as And-Inverter Graphs used in [21]. They consist of a total of 89 instances, subdivided into 3 groups: 29 instances encoding industrial Static Timing Analysis (STA) problems containing up to 13000 input variables, 40 instances from the "arithmetic" and "random_control" benchmarks of the EPFL suite [1] —combining the multiple outputs with an *or* or an *xor* operator— containing up to 1200 input variables, and 20 instances consisting of large randomly-generated AIGs containing up to 2800 input variables.

We notice that the discussed CNF encodings can be applied to AIGs as well, as an AIG can be seen as a non-CNF formula involving only $\wedge$ and $\neg$ operators. Moreover, we store formulas as DAGs, so that the same sub-formula is not duplicated multiple times, and the CNF encodings use the same Boolean atom to label the

different occurrences of the same sub-formula. Hence, all the discussed CNF encodings result in a CNF formula of linear size w.r.t. the size of the AIG.

The SMT($\mathcal{LRA}$) synthetic benchmarks. These problems were generated with the same procedure as the Boolean ones, with the difference that atoms are randomly-generated SMT($\mathcal{LRA}$) atoms over a set of R real variables $\{x_1, \ldots, x_R\}$, in the form $(\sum_{i=1}^{R} a_i x_i \leq b)$, where each a_i and b are random real coefficients. We generated 100 instances with depth $d = 5$, 100 instances with depth $d = 6$, and 100 instances with depth $d = 7$, all of them involving $R = 5$ real variables, for a total of 300 instances.

The WMI benchmarks. WMI problems were generated using the procedure described in [70]. Specifically, the paper addresses the problem of enumerating all the different paths of the weight function by encoding it into a skeleton formula. Each instance consists of a skeleton formula of a randomly-generated weight function, where the conditions are random formulas over Boolean and $\mathcal{LRA}$-atoms. Since the conditions are typically non-atomic, the resulting formula is not in CNF, and thus we preprocessed it with the different CNF-izations before enumerating its $\mathcal{TA}(\ldots)$. As done in [70], we fixed the number of Boolean atoms to 3, and the number of real variables to 3, and we generated 10 instances for each depth value 4, 5, 6, and 7, for a total of 40 problems.

5.3 Information for the Reproducibility of the Experiments

We ran MathSAT with the option -dpll.allsat_minimize_model=true to enumerate minimal partial assignments. For disjoint and non-disjoint enumeration, we set the options -dpll.allsat_allow_duplicates=false and -dpll.allsat_allow_duplicates=true, respectively. We also set the options -preprocessor.simplification=0 and -preprocessor.toplevel_propagation=false to disable several non-validity-preserving preprocessing steps. As discussed in §3.2 and §4, we also set the options -dpll.branching_initial_phase=0 and -dpll.branching_cache_phase=2 to split on the false branch first but enabling phase caching. TabularAllSAT was run with default options, which include branching on the false branch first. TabularAllSMT was run with the same options as MathSAT to disable preprocessing steps, since it uses MathSAT as a backend for theory reasoning.

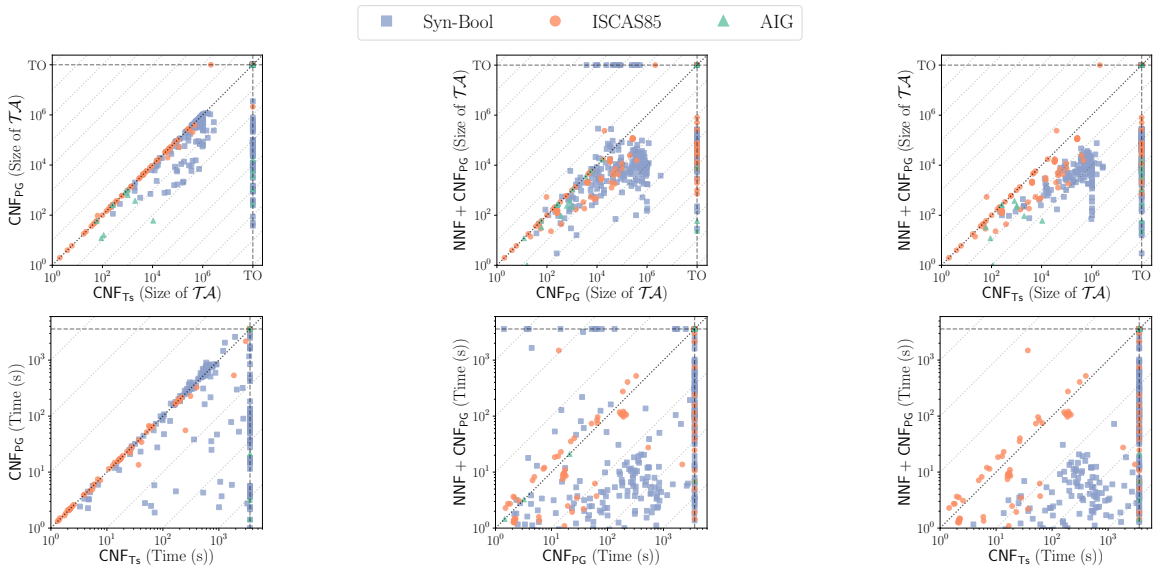
All the experiments were run on an Intel Xeon Gold 6238R @ 2.20GHz 28 Core machine with 128 GB of RAM, running Ubuntu Linux 20.04. For each problem set, we set a timeout of 3600s. Benchmarks, results, and source code are made available online on a Zenodo repository [50, 51]. An updated version of the source code is available at https://github.com/masinag/allsat-cnf.

5.4 Results

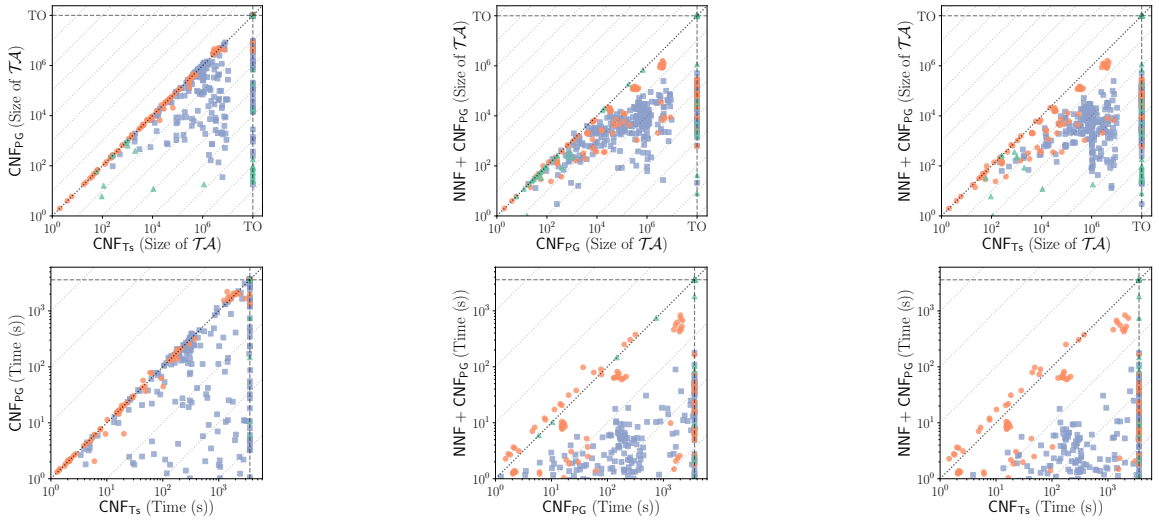
Figures 1 and 2 show the results for AllSAT with MathSAT and TabularAllSAT, respectively. Figures 3 and 4 show the results for AllSMT with MathSAT and TabularAllSMT, respectively. For each figure, we report a pair of subfigures comparing the CNF-izations for disjoint and non-disjoint enumeration (when both are supported). Each subfigure reports a set of scatter plots to compare $\mathsf{CNF}_{\mathsf{Ts}}$, $\mathsf{CNF}_{\mathsf{PG}}$ and NNF+$\mathsf{CNF}_{\mathsf{PG}}$ in terms of number of partial truth assignments (size of $\mathcal{TA}$), in the first row, and execution time, in the second row. Each problem set is represented by a different color and marker. (In §F, we report the scatter plots for each group of benchmarks separately.) Timeouts are represented by the points on the dashed line, and summarized in the tables below the plots.

Notice the logarithmic scale of the axes (!).

The Boolean synthetic benchmarks. (Figures 1 and 2) The plots show that in the disjoint case $\mathsf{CNF}_{\mathsf{PG}}$ performs better than $\mathsf{CNF}_{\mathsf{Ts}}$, since both MathSAT (Figure 1a) and TabularAllSAT (Figure 2a) enumerate a smaller $\mathcal{TA}$ (first row) in less time (second row) on every instance. Furthermore, the combination of NNF and $\mathsf{CNF}_{\mathsf{PG}}$ yields by far the best results for both solvers, drastically reducing the size of $\mathcal{TA}$ and the execution time by orders of



(a) Results for disjoint enumeration.

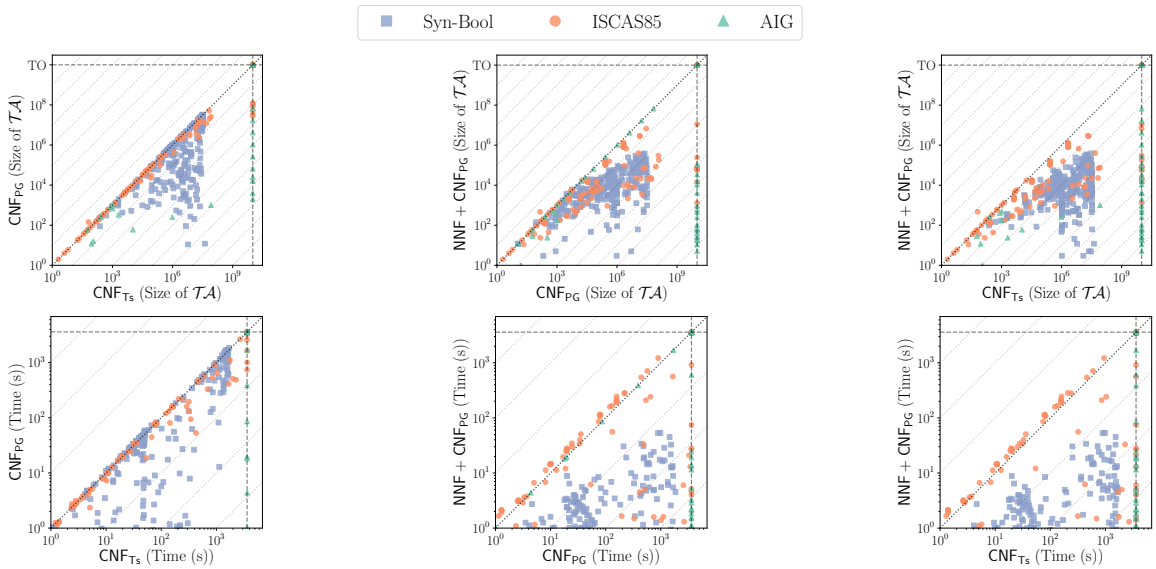


(b) Results for non-disjoint enumeration.

Bench.	Instances	T.O. for disjoint AllSAT			T.O. for non-disjoint AllSAT		
		CNF_{Ts}	CNF_{PG}	$\text{NNF+CNF}_{\text{PG}}$	CNF_{Ts}	CNF_{PG}	$\text{NNF+CNF}_{\text{PG}}$
Syn-Bool	300	151	84	**20**	88	48	**0**
ISCAS85	250	47	43	**30**	27	22	**1**
AIG	89	79	73	**71**	78	60	**50**

(c) Number of timeouts.

Fig. 1. Results on the Boolean benchmarks using MathSAT. Plots in 1a and 1b compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 1c. All axes use a logarithmic scale.



(a) Results for disjoint enumeration.

Bench.	Instances	T.O. for disjoint AllSAT		
		CNF_{Ts}	CNF_{PG}	$NNF+CNF_{PG}$
Syn-Bool	300	**0**	**0**	**0**
ISCAS85	250	17	11	**3**
AIG	89	77	67	**47**

(b) Number of timeouts.

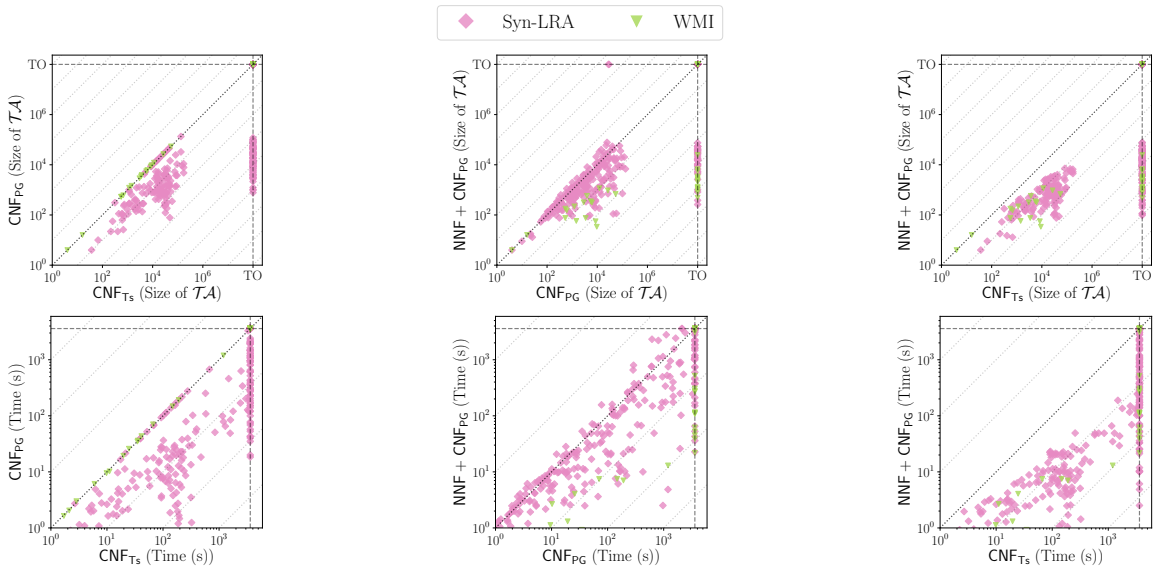
Fig. 2. Results on the Boolean benchmarks using TabularAllSAT. Plots in 2a compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 2b. All axes use a logarithmic scale.

magnitude w.r.t. both CNF_{Ts} and CNF_{PG}. The advantage of $NNF+CNF_{PG}$ over both CNF_{Ts} and CNF_{PG} is even more evident for the non-disjoint case (Figure 1b).

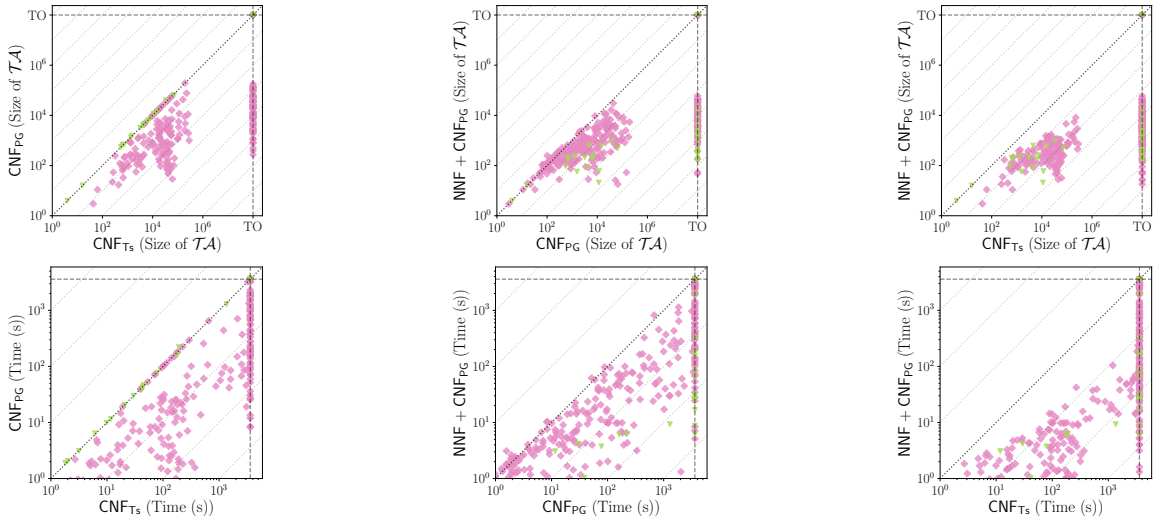
The ISCAS'85 benchmarks. (Figures 1 and 2) First, we notice that CNF_{Ts} and CNF_{PG} have very similar behavior, both in terms of execution time and size of $\mathcal{TA}$. The reason is that in circuits most of the sub-formulas occur with double polarity, so that the two encodings are very similar, if not identical.

Second, we notice that by converting the formula into NNF before applying CNF_{PG} the enumeration, both disjoint and non-disjoint, is much more effective, as a much smaller $\mathcal{TA}$ is enumerated, with only a few outliers. The fact that for some instances $NNF+CNF_{PG}$ takes a little more time can be due to the fact that it can produce a formula that is up to twice as large and contains up to twice as many label atoms as the other two encodings, increasing the time to find the assignments. Notice also that, even enumerating a smaller $\mathcal{TA}$ at a price of a small time overhead can be beneficial in many applications.

The AIG benchmarks. (Figures 1 and 2) This set of benchmarks is by far the most challenging one, as they contain many Boolean atoms and a very complex structure. For this reason, even the best-performing encoding, $NNF+CNF_{PG}$, reports many timeouts.



(a) Results for disjoint enumeration.

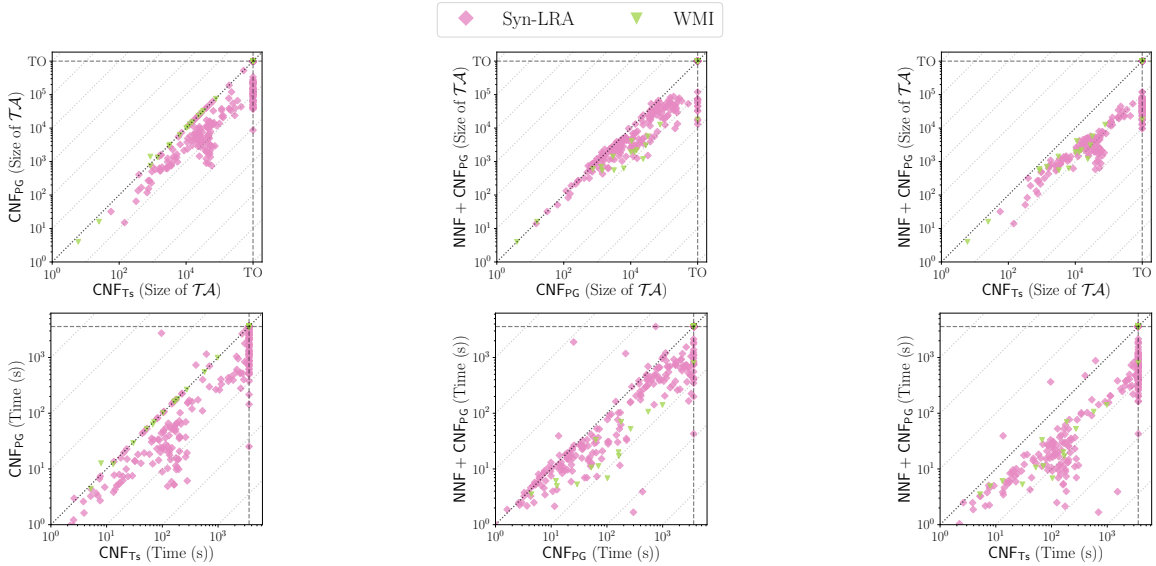


(b) Results for non-disjoint enumeration.

Bench.	Instances	T.O. for disjoint AllSMT			T.O. for non-disjoint AllSMT		
		CNF_{Ts}	CNF_{PG}	$\text{NNF+CNF}_{\text{PG}}$	CNF_{Ts}	CNF_{PG}	$\text{NNF+CNF}_{\text{PG}}$
Syn-LRA	300	155	88	**48**	152	74	**7**
WMI	40	23	23	**9**	23	23	**9**

(c) Number of timeouts.

Fig. 3. Results on the SMT($\mathcal{LRA}$) benchmarks using MathSAT. Plots in 3a, 3b compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 3c. All axes use a logarithmic scale.



(a) Results for disjoint enumeration.

Bench.	Instances	T.O. for disjoint AllSMT		
		$\mathsf{CNF}_{\mathsf{Ts}}$	$\mathsf{CNF}_{\mathsf{PG}}$	$\mathsf{NNF+CNF}_{\mathsf{PG}}$
Syn-LRA	300	147	92	**73**
WMI	40	24	24	**23**

(b) Number of timeouts.

Fig. 4. Results on the SMT($\mathcal{LRA}$) benchmarks using TabularAllSMT. Plots in 4a compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 4b. All axes use a logarithmic scale.

Nevertheless, we can still observe that the combination of NNF and $\mathsf{CNF}_{\mathsf{PG}}$ is the best-performing encoding for both solvers, both in terms of size of $\mathcal{TA}$ and execution time.

The SMT($\mathcal{LRA}$) synthetic benchmarks. (Figures 3 and 4) The plots confirm that the analysis holds also for the AllSMT case. The results are in line with those obtained on the Boolean benchmarks, for both the disjoint and non-disjoint cases.

The WMI benchmarks. (Figures 3 and 4) In these benchmarks, most of the sub-formulas occur with double polarity, so that $\mathsf{CNF}_{\mathsf{Ts}}$ and $\mathsf{CNF}_{\mathsf{PG}}$ encodings are almost identical, and they obtain very similar results in both metrics. The advantage is significant, instead, if the formula is converted into NNF upfront, since by using $\mathsf{NNF+CNF}_{\mathsf{PG}}$ the solvers enumerate a smaller $\mathcal{TA}$. In this application, it is crucial to enumerate as few partial truth assignments as possible, since for each truth assignment an integral must be computed, which is a very expensive operation [59, 60, 70, 71]. Notice also that WMI requires disjoint enumeration to decompose the whole integral into a sum of directly-computable integrals, one for each satisfying truth assignment. Nevertheless, for completeness, we also report the results for the non-disjoint case.

Overall Results. Overall, from the plots and tables in Figures 1 to 4, it is eye-catching that the usage of $\mathsf{NNF+CNF}_{\mathsf{PG}}$ instead of $\mathsf{CNF}_{\mathsf{Ts}}$ or $\mathsf{CNF}_{\mathsf{PG}}$ causes a dramatic improvement in the performances of MathSAT,

TabularAllSAT, and TabularAllSMT, in terms of both CPU time and size of the assignment sets generated, for both disjoint and non-disjoint enumeration, on all AllSAT and AllSMT benchmark sets under test.

5.5 Discussion: Using NNF+CNF_{PG} with Related Problems

We stress the fact that the proposed NNF+CNF_{PG} encoding is conceived and expected to be effective for *enumeration* techniques which generate *partial* assignments (see §4). Thus, we do not expect benefits in using it with plain SAT or SMT *solving* (see also Remark 1), nor with enumeration techniques which generate *total* or *nearly-total* assignments only. To this extent, we investigate empirically and discuss some related problems for which we do not expect benefits in using NNF+CNF_{PG} for the above reason. (See also §6.)

Plain SAT and SMT solving. To support Remark 1, in Appendix C we compare the different CNF encodings for plain SAT and SMT solving on all the benchmarks in §5.2. Even though these problems are very small for plain solving, and SAT and SMTsolvers deal with them efficiently, we can see that the usage of NNF+CNF_{PG} brings no advantage, and solving is uniformly slower than with CNF_{PG} or CNF_{Ts}. This shows that our novel technique works specifically for enumeration but not for solving, as expected.

Answer Set Programming. For the reason discussed above, we do not expect our new CNF encoding to bring benefits to Answer Set Programming (ASP), because answer sets represent *total* truth assignments, since they include atoms assigned to true, and other atoms are implicitly assumed to be assigned to false. (We received confirmation from the clasp and clingo developers that these tools do not support partial assignments. See the discussion at https://github.com/potassco/clingo/issues/512#issuecomment-2223162682.) Notice that the concept of "minimal" in ASP refers to the minimality of the set of atoms that are true in the answer set, and not to the minimality of the set of atoms that are assigned a truth value, as in this paper.

(Weighted) Model Counting. For (weighted) model counting the CNF_{Ts} encoding is currently the most suitable, because it preserves the model count, whereas CNF_{PG} —and hence NNF+CNF_{PG}— does not. In order to preserve the model count with these encodings and with current model counters, one should make them project away the **B** labels, drastically worsening their performances.

Although investigating ad-hoc model counting techniques for NNF+CNF_{PG}-encoded formulas (e.g., by exploiting the polarity of label variables) could be an interesting topic for future research, it would definitely exceed the scope of this paper.

d-DNNF compilation, (projected) enumeration and counting. We investigated the effect of the different CNFizations on d-DNNF compilation, (projected) enumeration, and model counting. We tested the tools d4+modelgraph [37] for projected d-DNNF compilation and enumeration, and d4 [38] for projected d-DNNF model counting, on the Boolean benchmarks in §5.2. Each formula was converted into CNF, and then compiled into d-DNNF projected on the original atoms **A**.

The results are reported in §D and §E for enumeration and model counting, respectively. We notice that, by using NNF+CNF_{PG} instead of CNF_{Ts} or CNF_{PG}, these tools (a) show no substantial difference in terms of number and size of the assignments generated, and (b) even worsen their time performance. The former fact (a) is due to the fact that d4_p algorithm [39] —implemented by d4 for projected compilation and counting— branches first on important atoms **A**, and only at the end of the branch it tries to assign also the non-important atoms **B** to be projected away. Consequently, the projected assignments include all or almost-all important atoms, thus producing total or nearly-total projected assignments on **A**. This is not surprising, since d-DNNF-based tools typically do not rely their efficiency on the enumeration of short partial assignments, but rather on the effective decomposition and caching of subproblems. The latter fact (b) may be due to several reasons: unlike with CNF_{Ts}, with CNF_{PG} the assignment of the **A** values does not force the deterministic assignment of the **B** values by BCP,

causing extra useless search; also, the NNF transformation up to doubles the number of the non-important atoms **B** to be projected away; furthermore, we conjecture that the duplication of labels B_i^+ and B_i^-, along with distinct encodings φ_i^+ and φ_i^- for $\mathsf{NNF}(\varphi_i)$ and $\mathsf{NNF}(\neg\varphi_i)$, may affect the effectiveness of the caching and partitioning mechanisms inside the D4 d-DNNF compiler.

Although in principle ad-hoc d-DNNF compilation strategies for d-DNNF-based model counting and enumeration could be devised to exploit the properties of our $\mathsf{NNF{+}CNF}_{\mathsf{PG}}$ encoding, investigating such techniques would exceed the scope of this paper.

6 Related Work

Applications of AllSAT and AllSMT. SAT and SMT enumeration has an important role in a variety of applications, ranging from artificial intelligence to formal verification. AllSAT and AllSMT, mainly in their disjoint version, play a foundational role in several frameworks for *probabilistic reasoning*, such as model counting in SMT (*#SMT*) [14] and *Weighted Model Integration (WMI)* [5, 59, 60, 70, 71]. Specifically, #SMT($\mathcal{LRA}$) [49, 81, 23] consists in summing up the volumes of the convex polytopes defined by each of the $\mathcal{LRA}$-satisfiable truth assignments propositionally satisfying a SMT($\mathcal{LRA}$) formula, and has been employed for value estimation of probabilistic programs [14] and for quantitative program analysis [45]. WMI can be seen as a generalization of #SMT($\mathcal{LRA}$) that additionally considers a weight function w that has to be integrated over each of such polytopes, and has been used to perform inference in hybrid probabilistic models such as Bayesian and Markov networks [5] and Density Estimation Trees [70]. AllSAT, both disjoint and non-disjoint, has applications also in *data mining*, where the problem of frequent itemsets can be encoded into a propositional formula whose satisfying assignments are the frequent itemsets [10, 19]. It has also been used in the context of *software testing* to generate a suite of test inputs that should match a given output [35], and in *circuit design*, to convert a formula from CNF to DNF [55, 54, 6], and for Static Timing Analysis to determine the inputs that can trigger a timing violation in a circuit [21]. AllSAT and AllSMT have also been applied in *network verification* for checking network reachability and for analyzing the correctness and consistency of network connectivity policies [47, 32, 46]. Moreover, they have been used for computing the image and preimage of a given set of states in *unbounded model checking* [53, 26, 42], and they are also at the core of algorithms for computing *predicate abstraction*, a concept widely used in formal verification for automatically computing finite-state abstractions for systems with potentially infinite state space [40, 17, 41].

AllSAT. Most of the works on AllSAT have focused on the enumeration of satisfying assignments for CNF formulas (e.g., [61, 28, 79, 77, 76, 43, 72]), with several efforts in developing efficient and effective techniques for minimizing partial assignments (e.g., [67, 61, 76]). The problem of minimizing truth assignments for Tseitin-encoded problems was addressed in [29]. They propose to make iterative calls to a SAT solver imposing increasingly tighter cardinality constraints to obtain a minimal assignment. Whereas this approach can be used to find a single short truth assignment, it can be very expensive, and thus it is unsuitable for enumeration.

Other works have concentrated on the enumeration of satisfying assignments for combinatorial circuits, exploiting the structural information of the circuits to minimize the partial assignments over the input variables (e.g., [33, 34, 74, 21, 20]). In [37], the authors proposed a tool to enumerate disjoint partial satisfying assignments of formulas in decomposable, deterministic NNF (d-DNNF). The tool takes as input a d-DNNF formula compiled with D4 [38], and then efficiently traverses it to enumerate the partial assignments. However, projected d-DNNFs typically do not allow for finding short partial assignments [39], and hence preprocessing the formula with our encoding does not bring any benefit, as confirmed by the experimental evaluation in §5.5.

A problem closely related to AllSAT is that of finding all the prime implicants of a formula (e.g., [66, 30, 48]). AllSAT is a much simpler problem, and can be viewed as the problem of finding a subset of implicants (i.e., partial satisfying assignments), not necessarily prime (i.e., minimal), which covers all the total satisfying assignments of the formula.

Projected AllSAT. Projected enumeration, i.e., enumeration of satisfying assignments over a set of relevant atoms, has been studied mainly for CNF formulas, e.g., in [26, 42, 41, 75, 73]. The ambiguity of the notion of "satisfiability by partial assignment" for non-CNF and existentially-quantified formulas has been raised in [68, 57, 69], highlighting the difference between "evaluation to true", which is simpler to check and typically used by SAT solvers, and "logical entailment", which allows for producing shorter assignments. The approach based on dual reasoning in [56, 58], although able to detect logical entailment and thus to produce shorter partial assignments, is very inefficient even for small formulas.

AllSMT. The literature on AllSMT is very limited, and AllSMT algorithms are highly based on AllSAT techniques and tools. For instance, MathSAT5 [15] implements an AllSMT functionality based on the procedure in [41]. A similar procedure has been described in [63]. An AllSMT algorithm based on chronological backtracking has been recently proposed in [73].

Enumeration in related areas. In Answer Set Programming (ASP), answer sets can be seen as truth assignments satisfying a logic program. ASP programs can be translated into CNF formulas and vice versa, so as to have a one-to-one correspondence between satisfying assignments and answer sets [16, 44]. Answer sets, however, correspond to *total* truth assignments. Thus, our work is not directly applicable to ASP.

Propositional Model Counting (#SAT) is the task of *counting* the number of satisfying assignments of a propositional formula. Counting is simpler than enumeration, since only the number of truth assignments is relevant, but a complete exploration of the space of solutions is still needed. (This is different from #SMT, where each truth assignment is "consumed" to compute a volume.) Efficient algorithms have been developed for #SAT on CNF formulas, mostly based either on DPLL-like exhaustive search, or on Knowledge Compilation (see [25]). Notably, [2] proposed the model counter #clasp based on enumeration of minimal partial assignment, which would likely benefit from our encoding. Unfortunately, we did not receive an answer from the authors about the availability of the tool.

The role of CNF encodings. Although most AllSAT solvers assume the formulas to be in CNF, little or no work has been done to investigate the impact of the different CNF encodings on their effectiveness and efficiency. The role of the CNF-ization has been widely studied for SAT solving (e.g., [11, 31, 9, 36]) and in a recent work also for propositional model counting in the context of feature model analysis [36].

The idea of using two different variables to label the positive and negative occurrences of a sub-formula shares some similarities with the so-called *dual-rail* encoding [13, 62], which has been shown to be successful for prime implicant enumeration [66, 48], and recently also for AllSAT on combinatorial circuits [21].

Given a CNF formula, the dual-rail encoding maps each atom A_i into a pair of dual-rail atoms $\langle A_i^+, A_i^- \rangle$, s.t. $A_i, \neg A_i$ are substituted with A_i^+, A_i^-, respectively, and $(\neg A_i^+ \vee \neg A_i^-)$ is conjoined with the formula. The resulting CNF formula is equisatisfiable to the original one, and every total assignment η satisfying the dual-rail encoding corresponds to a partial assignment μ satisfying the original formula. Such μ assigns $\mu(A_i)$ to $\top$ or $\bot$ if the pair $\langle \eta(A_i^+), \eta(A_i^-) \rangle$ is $\langle \top, \bot \rangle$ or $\langle \bot, \top \rangle$, respectively, and leaves it unassigned if $\langle \eta(A_i^+), \eta(A_i^-) \rangle = \langle \bot, \bot \rangle$. Short partial assignments can be obtained by maximizing the number of pairs assigned to $\langle \bot, \bot \rangle$, which can be done either exactly by solving a MaxSAT problem, or heuristically by assigning negative value first to decision atoms [21].

Using the dual-rail encoding for enumeration, however, requires ad-hoc enumeration algorithms, since the solver must take into account the three-valued semantics of truth assignments over dual-rail atoms when enumerating the assignments [21]. Our contribution, instead, focuses on CNF-ization approaches that can be used in combination with any enumeration algorithm matching the properties described in §2.2. Also, comparing the $\mathsf{NNF} + \mathsf{CNF}_{\mathsf{PG}}$ encoding with the dual-rail encoding, we observe that the former duplicates only the label atoms in $\mathbf{B}$, introducing fewer atoms than the dual-rail encoding. Moreover, because of this, the clauses in the

form $(\neg B_i^+ \vee \neg B_i^-)$ are not needed for correctness but only for efficiency, and can in principle be omitted. In the dual-rail encoding, instead, their presence is essential to prevent the illegal assignment $\eta(B_i^+) = \eta(B_i^-) = \top$.

7 Conclusions and Future Work

We have presented a theoretical and empirical analysis of the impact of different CNF-ization approaches on SAT and SMT enumeration, both disjoint and non-disjoint. We have shown how the most popular transformations conceived for SAT and SMT solving, namely the Tseitin and the Plaisted and Greenbaum CNF-izations, prevent the solver from producing short partial assignments, thus seriously affecting the effectiveness of the enumeration. To overcome this limitation, we have proposed to preprocess the formula by converting it into NNF before applying the Plaisted and Greenbaum transformation. We have shown, both theoretically and empirically, that the latter approach can fully overcome the problem and can drastically reduce both the number of partial assignments and the execution time.

This work opens an interesting research avenue: investigate the role of CNF-ization in neighbor fields as d-DNNF compilation and model counting, possibly adapting d-DNNF compilers and model counters so that to exploit different forms of CNF-izations.

Acknowledgments

GS and RS were partially supported by the project "AI@TN" funded by the Autonomous Province of Trento. RS was partially supported by the MUR PNRR project FAIR - Future AI Research (PE00000013) funded by the NextGenerationEU. RS was partially funded under the NRRP, Mission 4 Component 2 Investment 1.4, by the European Union — NextGenerationEU (proj. nr. CN 00000013). RS was supported in part by the TANGO project funded by the EU Horizon Europe research and innovation program under GA No 101120763, funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union, the European Health and Digital Executive Agency (HaDEA) or The European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Luca Amarù, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL Combinational Benchmark Suite. In *Proceedings of the 24th International Workshop on Logic & Synthesis*.

[2] Rehan Abdul Aziz, Geoffrey Chu, Christian Muise, and Peter Stuckey. 2015. #∃SAT: Projected Model Counting. In *Theory and Applications of Satisfiability Testing – SAT 2015*. Springer International Publishing, 121–137. https://doi.org/10.1007/978-3-319-24318-4_10

[3] Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. 2021. Satisfiability Modulo Theories. In *Handbook of Satisfiability* (2 ed.). Frontiers in Artificial Intelligence and Applications, Vol. 336. IOS Press, 1267–1329. https://doi.org/10.3233/FAIA201017

[4] Roberto J. Bayardo and Robert C. Schrag. 1997. Using CSP Look-Back Techniques to Solve Real-World SAT Instances. In *Proceedings of the 14th AAAI Conference on Artificial Intelligence (AAAI'97)*. AAAI Press, 203–208.

[5] Vaishak Belle, Andrea Passerini, and Guy Van den Broeck. 2015. Probabilistic Inference in Hybrid Domains by Weighted Model Integration. In *24th International Joint Conference on Artificial Intelligence*. AAAI Press, 2770–2776.

[6] Anna Bernasconi, Valentina Ciriani, Fabrizio Luccio, and Linda Pagli. 2013. Compact DSOP and Partial DSOP Forms. *Theory of Computing Systems* 53, 4 (Nov. 2013), 583–608. https://doi.org/10.1007/s00224-013-9447-2

[7] Armin Biere. 2008. PicoSAT Essentials. *Journal on Satisfiability, Boolean Modeling and Computation* 4, 2-4 (May 2008), 75–97. https://doi.org/10.3233/SAT190039

[8] A. Biere, M. Heule, and H. van Maaren. 2021. *Handbook of Satisfiability* (2 ed.). Frontiers in Artificial Intelligence and Applications, Vol. 336. IOS Press. https://doi.org/10.3233/FAIA336
[9] Magnus Björk. 2009. Successful SAT Encoding Techniques. *Journal on Satisfiability, Boolean Modeling and Computation* 7, 4 (July 2009), 189–201. https://doi.org/10.3233/SAT190085
[10] Abdelhamid Boudane, Said Jabbour, Lakhdar Sais, and Yakoub Salhi. 2016. A SAT-based Approach for Mining Association Rules. In *25th International Joint Conference on Artificial Intelligence*. AAAI Press, 2472–2478.
[11] Thierry Boy de la Tour. 1992. An Optimality Result for Clause Form Translation. *Journal of Symbolic Computation* 14, 4 (Oct. 1992), 283–301. https://doi.org/10.1016/0747-7171(92)90009-S
[12] F. Brglez and H. Fujiwara. 1985. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. In *Proceedings of IEEE International Symposium Circuits and Systems (ISCAS 85)*. IEEE Press, 677–692.
[13] Randal E. Bryant, D. Beatty, K. Brace, K. Cho, and T. Sheffler. 1987. COSMOS: A Compiled Simulator for MOS Circuits. In *Proceedings of the 24th ACM/IEEE Design Automation Conference (DAC '87)*. Association for Computing Machinery, 9–16. https://doi.org/10.1145/37888.37890
[14] Dmitry Chistikov, Rayna Dimitrova, and Rupak Majumdar. 2017. Approximate Counting in SMT and Value Estimation for Probabilistic Programs. *Acta Informatica* 54, 8 (Dec. 2017), 729–764. https://doi.org/10.1007/s00236-017-0297-2
[15] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. 2013. The MathSAT5 SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems (Lecture Notes in Computer Science)*. Springer, 93–107. https://doi.org/10.1007/978-3-642-36742-7_7
[16] Keith L. Clark. 1978. Negation as Failure. In *Logic and Data Bases*, Hervé Gallaire and Jack Minker (Eds.). Springer US, 293–322. https://doi.org/10.1007/978-1-4684-3384-5_11
[17] Edmund Clarke, Daniel Kroening, Natasha Sharygina, and Karen Yorav. 2004. Predicate Abstraction of ANSI-C Programs Using SAT. *Formal Methods in System Design* 25, 2 (Sept. 2004), 105–127. https://doi.org/10.1023/B:FORM.0000040025.89719.f3
[18] Adnan Darwiche and Pierre Marquis. 2002. A Knowledge Compilation Map. *Journal of Artificial Intelligence Research* 17, 1 (Sept. 2002), 229–264.
[19] Imen Ouled Dlala, Said Jabbour, Lakhdar Sais, and Boutheina Ben Yaghlane. 2016. A Comparative Study of SAT-Based Itemsets Mining. In *Research and Development in Intelligent Systems XXXIII*. Springer International Publishing, 37–52. https://doi.org/10.1007/978-3-319-47175-4_3
[20] Dror Fried, Alexander Nadel, Roberto Sebastiani, and Yogev Shalmon. 2024. Entailing Generalization Boosts Enumeration. In *27th International Conference on Theory and Applications of Satisfiability Testing (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 305)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 13:1–13:14. https://doi.org/10.4230/LIPIcs.SAT.2024.13
[21] Dror Fried, Alexander Nadel, and Yogev Shalmon. 2023. AllSAT for Combinational Circuits. In *26th International Conference on Theory and Applications of Satisfiability Testing (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 271)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 9:1–9:18. https://doi.org/10.4230/LIPIcs.SAT.2023.9
[22] Marco Gario and Andrea Micheli. 2015. PySMT: A Solver-Agnostic Library for Fast Prototyping of SMT-based Algorithms. In *SMT Workshop 2015*.
[23] Cunjing Ge, Feifei Ma, Peng Zhang, and Jian Zhang. 2018. Computing and Estimating the Volume of the Solution Space of SMT(LA) Constraints. *Theoretical Computer Science* 743 (Sept. 2018), 110–129. https://doi.org/10.1016/j.tcs.2016.10.019
[24] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. 2007. Conflict-Driven Answer Set Enumeration. In *Logic Programming and Nonmonotonic Reasoning (Lecture Notes in Computer Science, Vol. 4483)*. Springer Berlin Heidelberg, 136–148. https://doi.org/10.1007/978-3-540-72200-7_13

[25] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. 2021. Model Counting. In *Handbook of Satisfiability*. IOS Press, 993–1014. https://doi.org/10.3233/FAIA201009

[26] Orna Grumberg, Assaf Schuster, and Avi Yadgar. 2004. Memory Efficient All-Solutions SAT Solver and Its Application for Reachability Analysis. In *Formal Methods in Computer Aided Design*. Vol. 3312. Springer Berlin Heidelberg, 275–289. https://doi.org/10.1007/978-3-540-30494-4_20

[27] M.C. Hansen, H. Yalcin, and J.P. Hayes. 1999. Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering. *IEEE Design & Test of Computers* 16, 3 (July 1999), 72–80. https://doi.org/10.1109/54.785838

[28] Jinbo Huang and Adnan Darwiche. 2004. Using DPLL for Efficient OBDD Construction. In *7th International Conference on Theory and Applications of Satisfiability Testing (Lecture Notes in Computer Science)*. Springer-Verlag, 157–172. https://doi.org/10.1007/11527695_13

[29] Markus Iser, Carsten Sinz, and Mana Taghdiri. 2013. Minimizing Models for Tseitin-Encoded SAT Instances. In *16th International Conference on Theory and Applications of Satisfiability Testing (Lecture Notes in Computer Science)*. Springer-Verlag, 224–232. https://doi.org/10.1007/978-3-642-39071-5_17

[30] Said Jabbour, Joao Marques-Silva, Lakhdar Sais, and Yakoub Salhi. 2014. Enumerating Prime Implicants of Propositional Formulae in Conjunctive Normal Form. In *Logics in Artificial Intelligence (Lecture Notes in Computer Science)*. Springer International Publishing, 152–165. https://doi.org/10.1007/978-3-319-11558-0_11

[31] Paul Jackson and Daniel Sheridan. 2005. Clause Form Conversions for Boolean Circuits. In *7th International Conference on Theory and Applications of Satisfiability Testing (Lecture Notes in Computer Science)*. Springer, 183–198. https://doi.org/10.1007/11527695_15

[32] Karthick Jayaraman, Nikolaj Bjørner, Geoff Outhred, and Charlie Kaufman. 2014. *Automated Analysis and Debugging of Network Connectivity Policies*. Technical Report MSR-TR-2014-102. Microsoft.

[33] HoonSang Jin, HyoJung Han, and Fabio Somenzi. 2005. Efficient Conflict Analysis for Finding All Satisfying Assignments of a Boolean Circuit. In *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 3440. Springer Berlin Heidelberg, 287–300. https://doi.org/10.1007/978-3-540-31980-1_19

[34] HoonSang Jin and Fabio Somenzi. 2005. Prime Clauses for Fast Enumeration of Satisfying Assignments to Boolean Circuits. In *Proceedings of the 42nd Annual Design Automation Conference*. 750–753.

[35] Sarfraz Khurshid, Darko Marinov, Ilya Shlyakhter, and Daniel Jackson. 2004. A Case for Efficient Solution Enumeration. In *7th International Conference on Theory and Applications of Satisfiability Testing (Lecture Notes in Computer Science)*. Springer, 272–286. https://doi.org/10.1007/978-3-540-24605-3_21

[36] Elias Kuiter, Sebastian Krieter, Chico Sundermann, Thomas Thüm, and Gunter Saake. 2022. Tseitin or Not Tseitin? The Impact of CNF Transformations on Feature-Model Analyses. In *37th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 1–13. https://doi.org/10.1145/3551349.3556938

[37] Jean-Marie Lagniez and Emmanuel Lonca. 2024. Leveraging Decision-DNNF Compilation for Enumerating Disjoint Partial Models. In *21st International Conference on Principles of Knowledge Representation and Reasoning*.

[38] Jean-Marie Lagniez and Pierre Marquis. 2017. An Improved Decision-DNNF Compiler. In *26th International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 667–673. https://doi.org/10.24963/ijcai.2017/93

[39] Jean-Marie Lagniez and Pierre Marquis. 2019. A Recursive Algorithm for Projected Model Counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 1536–1543. https://doi.org/10.1609/aaai.v33i01.33011536

[40] Shuvendu K. Lahiri, Randal E. Bryant, and Byron Cook. 2003. A Symbolic Approach to Predicate Abstraction. In *Computer Aided Verification (Lecture Notes in Computer Science)*. Springer, 141–153. https://doi.org/10.1007/978-3-540-45069-6_15

[41] Shuvendu K. Lahiri, Robert Nieuwenhuis, and Albert Oliveras. 2006. SMT Techniques for Fast Predicate Abstraction. In *Computer Aided Verification (Lecture Notes in Computer Science)*. Springer-Verlag, 424–437. https://doi.org/10.1007/11817963_39

[42] Bin Li, M.S. Hsiao, and Shuo Sheng. 2004. A Novel SAT All-Solutions Solver for Efficient Preimage Computation. In *Automation and Test in Europe Conference and Exhibition Proceedings Design*, Vol. 1. 272–277. https://doi.org/10.1109/DATE.2004.1268860

[43] Jiaxin Liang, Feifei Ma, Junping Zhou, and Minghao Yin. 2022. AllSATCC: Boosting AllSAT Solving with Efficient Component Analysis. In *31st International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 1866–1872. https://doi.org/10.24963/ijcai.2022/259

[44] Fangzhen Lin and Yuting Zhao. 2004. ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. *Artificial Intelligence* 157, 1 (Aug. 2004), 115–137. https://doi.org/10.1016/j.artint.2004.04.004

[45] Sheng Liu and Jian Zhang. 2011. Program Analysis: From Qualitative Analysis to Quantitative Analysis (NIER Track). In *33rd International Conference on Software Engineering*. 956–959. https://doi.org/10.1145/1985793.1985957

[46] Nuno P. Lopes, Nikolaj Bjørner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. 2015. Checking Beliefs in Dynamic Networks. In *12th USENIX Symposium on Networked Systems Design and Implementation*. 499–512.

[47] Nuno P. Lopes, Nikolaj Bjørner, Patrice Godefroid, and George Varghese. 2013. Network Verification in the Light of Program Verification. *MSR* Rep (Sept. 2013).

[48] Weilin Luo, Hai Want, Hongzhen Zhong, Ou Wei, Biqing Fang, and Xiaotong Song. 2021. An Efficient Two-phase Method for Prime Compilation of Non-clausal Boolean Formulae. In *2021 IEEE/ACM International Conference On Computer Aided Design*. 1–9. https://doi.org/10.1109/ICCAD51958.2021.9643520

[49] Feifei Ma, Sheng Liu, and Jian Zhang. 2009. Volume Computation for Boolean Combination of Linear Arithmetic Constraints. In *Automated Deduction – CADE-22 (Lecture Notes in Computer Science)*. Springer, 453–468. https://doi.org/10.1007/978-3-642-02959-2_33

[50] Gabriele Masina. 2024. On CNF Conversion for SAT and SMT Enumeration: Benchmarks, Results and Plots. Zenodo. https://doi.org/10.5281/zenodo.14035594.

[51] Gabriele Masina. 2024. On CNF Conversion for SAT and SMT Enumeration: Source Code. Zenodo. https://doi.org/10.5281/zenodo.14033421 https://doi.org/10.5281/zenodo.14033421.

[52] Gabriele Masina, Giuseppe Spallitta, and Roberto Sebastiani. 2023. On CNF Conversion for Disjoint SAT Enumeration. In *26th International Conference on Theory and Applications of Satisfiability Testing (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 271)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 15:1–15:16. https://doi.org/10.4230/LIPIcs.SAT.2023.15

[53] Kenneth L. McMillan. 2002. Applying SAT Methods in Unbounded Symbolic Model Checking. In *Computer Aided Verification (Lecture Notes in Computer Science)*. Springer-Verlag, 250–264. https://doi.org/10.1007/3-540-45657-0_19

[54] Peter Bro Miltersen, Jaikumar Radhakrishnan, and Ingo Wegener. 2005. On Converting CNF to DNF. *Theoretical Computer Science* 347, 1-2 (Nov. 2005), 325–335. https://doi.org/10.1016/j.tcs.2005.07.029

[55] S. Minato and G. De Micheli. 1998. Finding All Simple Disjunctive Decompositions Using Irredundant Sum-of-Products Forms. In *1998 IEEE/ACM International Conference on Computer-Aided Design*. 111–117. https://doi.org/10.1145/288548.288586

[56] Sibylle Möhle and Armin Biere. 2018. Dualizing Projected Model Counting. In *30th IEEE International Conference on Tools with Artificial Intelligence*. 702–709. https://doi.org/10.1109/ICTAI.2018.00111

[57] Sibylle Möhle, Roberto Sebastiani, and Armin Biere. 2020. Four Flavors of Entailment. In *23rd International Conference on Theory and Applications of Satisfiability Testing (Lecture Notes in Computer Science)*. Springer International Publishing, 62–71. https://doi.org/10.1007/978-3-030-51825-7_5

[58] Sibylle Möhle, Roberto Sebastiani, and Armin Biere. 2025. On Enumerating Short Projected Models. *Discrete Applied Mathematics* 361 (Jan. 2025), 412–439. https://doi.org/10.1016/j.dam.2024.10.021
[59] Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. 2017. Efficient Weighted Model Integration via SMT-Based Predicate Abstraction. In *26th International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 720–728. https://doi.org/10.24963/ijcai.2017/100
[60] Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. 2019. Advanced SMT Techniques for Weighted Model Integration. *Artificial Intelligence* 275, C (2019), 1–27. https://doi.org/10.1016/j.artint.2019.04.003
[61] Antonio Morgado and Joao Marques-Silva. 2005. Good Learning and Implicit Model Enumeration. In *17th IEEE International Conference on Tools with Artificial Intelligence*. IEEE Computer Society, 131–136.
[62] Luigi Palopoli, Fiora Pirri, and Clara Pizzuti. 1999. Algorithms for Selective Enumeration of Prime Implicants. *Artificial Intelligence* 111, 1-2 (July 1999), 41–72. https://doi.org/10.1016/S0004-3702(99)00035-1
[63] Q. Phan and P. Malacaria. 2015. All-Solution Satisfiability Modulo Theories: Applications, Algorithms and Benchmarks. In *10th International Conference on Availability, Reliability and Security*. IEEE, 100–109. https://doi.org/10.1109/ARES.2015.14
[64] David A. Plaisted and Steven Greenbaum. 1986. A Structure-preserving Clause Form Translation. *Journal of Symbolic Computation* 2, 3 (1986), 293–304. https://doi.org/10.1016/S0747-7171(86)80028-1
[65] Steven Prestwich. 2021. CNF Encodings. In *Handbook of Satisfiability* (2 ed.). Frontiers in Artificial Intelligence and Applications, Vol. 336. IOS Press, 75–100. https://doi.org/10.3233/FAIA200985
[66] Alessandro Previti, Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. 2015. Prime Compilation of Non-Clausal Formulae. In *24th International Joint Conference on Artificial Intelligence*.
[67] Kavita Ravi and Fabio Somenzi. 2004. Minimal Assignments for Bounded Model Checking. In *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 2988. Springer Berlin Heidelberg, 31–45. https://doi.org/10.1007/978-3-540-24730-2_3
[68] Roberto Sebastiani. 2020. Are You Satisfied by This Partial Assignment? arXiv preprint arXiv:2003.04225. https://doi.org/10.48550/arXiv.2003.04225 arXiv:2003.04225 [cs]
[69] Roberto Sebastiani. 2025. Entailment vs. Verification for Partial-assignment Satisfiability and Enumeration. In *Automated Deduction – CADE-30 (Lecture Notes in Computer Science)*. Springer. To appear. Available also as arXiv preprint https://arxiv.org/abs/2503.01536.
[70] Giuseppe Spallitta, Gabriele Masina, Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. 2022. SMT-based Weighted Model Integration with Structure Awareness. In *38th Conference on Uncertainty in Artificial Intelligence*, Vol. 180. PMLR, 1876–1885.
[71] Giuseppe Spallitta, Gabriele Masina, Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. 2024. Enhancing SMT-based Weighted Model Integration by Structure Awareness. *Artificial Intelligence* 328 (March 2024), 104067. https://doi.org/10.1016/j.artint.2024.104067
[72] Giuseppe Spallitta, Roberto Sebastiani, and Armin Biere. 2024. Disjoint Partial Enumeration without Blocking Clauses. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 8126–8135. https://doi.org/10.1609/aaai.v38i8.28652
[73] Giuseppe Spallitta, Roberto Sebastiani, and Armin Biere. 2025. Disjoint Projected Enumeration for SAT and SMT without Blocking Clauses. *Artificial Intelligence* 345 (2025), 104346.
[74] Abraham Temesgen Tibebu and Goerschwin Fey. 2018. Augmenting All Solution SAT Solving for Circuits with Structural Information. In *IEEE 21st International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. 117–122. https://doi.org/10.1109/DDECS.2018.00028
[75] Takahisa Toda and Takeru Inoue. 2017. Exploiting Functional Dependencies of Variables in All Solutions SAT Solvers. *Journal of Information Processing* 25, 0 (2017), 459–468. https://doi.org/10.2197/ipsjjip.25.459
[76] Takahisa Toda and Takehide Soh. 2016. Implementing Efficient All Solutions SAT Solvers. *ACM Journal of Experimental Algorithmics* 21 (Nov. 2016), 1–44. https://doi.org/10.1145/2975585

[77] Takahisa Toda and Koji Tsuda. 2015. BDD Construction for All Solutions SAT and Efficient Caching Mechanism. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 1880–1886. https://doi.org/10.1145/2695664.2695941

[78] Grigori S. Tseitin. 1983. On the Complexity of Derivation in Propositional Calculus. In *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*. Springer, 466–483. https://doi.org/10.1007/978-3-642-81955-1_28

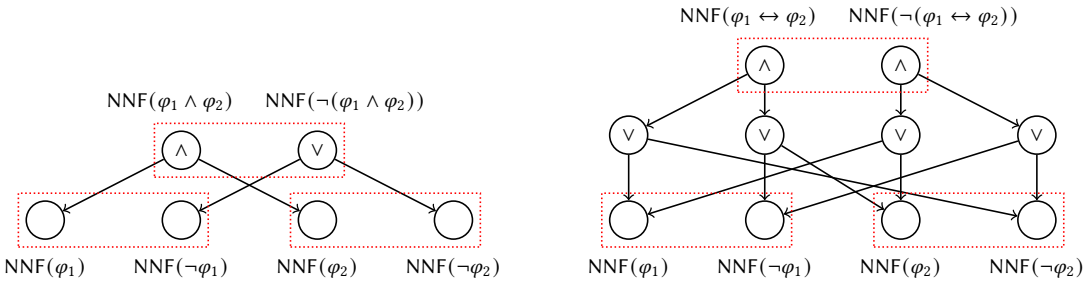
[79] Yinlei Yu, Pramod Subramanyan, Nestan Tsiskaridze, and Sharad Malik. 2014. All-SAT Using Minimal Blocking Clauses. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*. 86–91. https://doi.org/10.1109/VLSID.2014.22

[80] Yueling Zhang, Geguang Pu, and Jun Sun. 2020. Accelerating All-SAT Computation with Short Blocking Clauses. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 6–17. https://doi.org/10.1145/3324884.3416569

[81] Min Zhou, Fei He, Xiaoyu Song, Shi He, Gangyi Chen, and Ming Gu. 2015. Estimating the Volume of Solution Space for Satisfiability Modulo Linear Real Arithmetic. *Theory of Computing Systems* 56, 2 (Feb. 2015), 347–371. https://doi.org/10.1007/s00224-014-9553-9

A Proofs

A.1 Proof for Lemma 1 in Section 2.1



(a) Case $\varphi \stackrel{\text{def}}{=} \varphi_1 \wedge \varphi_2$. (b) Case $\varphi \stackrel{\text{def}}{=} \varphi_1 \leftrightarrow \varphi_2$.

Fig. 5. 2-root DAGs for the pair $\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle$ for $\varphi \stackrel{\text{def}}{=} \varphi_1 \wedge \varphi_2$ and $\varphi \stackrel{\text{def}}{=} \varphi_1 \leftrightarrow \varphi_2$ (those for $\varphi_1 \vee \varphi_2$ and $\varphi_1 \rightarrow \varphi_2$ are similar to that of $\varphi_1 \wedge \varphi_2$).

Proof. $\mathsf{NNF}(\varphi)$ is a sub-graph of the 2-root DAG for the pair $\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle$. We prove that the latter grows linearly in size w.r.t. φ by reasoning inductively on the structure of φ. (The size of a DAG is denoted with "$|\ldots|$".)

if φ is an atom: $\mathsf{NNF}(\varphi) = \varphi$ and $\mathsf{NNF}(\neg\varphi) = \neg\varphi$, so that: $|\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle| = 2$.

if $\varphi \stackrel{\text{def}}{=} \neg\varphi_1$: we assume by induction that $|\langle \mathsf{NNF}(\varphi_1), \mathsf{NNF}(\neg\varphi_1)\rangle|$ is linear in $|\varphi_1|$.
Then $\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle = \langle \mathsf{NNF}(\neg\varphi_1), \mathsf{NNF}(\varphi_1)\rangle$ (i.e., we just invert the order), so that

$$|\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle| = |\langle \mathsf{NNF}(\neg\varphi_1), \mathsf{NNF}(\varphi_1)\rangle| = |\langle \mathsf{NNF}(\varphi_1), \mathsf{NNF}(\neg\varphi_1)\rangle| \tag{12}$$

if $\varphi \stackrel{\text{def}}{=} (\varphi_1 \bowtie \varphi_2)$ s.t. $\bowtie \in \{\wedge, \leftrightarrow\}$: we assume by induction that $|\langle \mathsf{NNF}(\varphi_1), \mathsf{NNF}(\neg\varphi_1)\rangle|$ and $|\langle \mathsf{NNF}(\varphi_2), \mathsf{NNF}(\neg\varphi_2)\rangle|$ are linear in $|\varphi_1|$ and $|\varphi_2|$, respectively. (See Figure 5):

if $\bowtie$ is $\wedge$: the DAGs for $\mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)$ add 2 "$\wedge/\vee$" nodes and 2 + 2 arcs:

$\mathsf{NNF}(\quad \varphi_1 \wedge \varphi_2\ ) \stackrel{\text{def}}{=} \mathsf{NNF}(\quad \varphi_1) \wedge \mathsf{NNF}(\quad \varphi_2)$ and

$\mathsf{NNF}(\neg(\varphi_1 \wedge \varphi_2)) \stackrel{\text{def}}{=} \mathsf{NNF}(\neg\varphi_1) \vee \mathsf{NNF}(\neg\varphi_2)$, thus

$$|\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle| = 6 + |\langle \mathsf{NNF}(\varphi_1), \mathsf{NNF}(\neg\varphi_1)\rangle| + |\langle \mathsf{NNF}(\varphi_2), \mathsf{NNF}(\neg\varphi_2)\rangle| \tag{13}$$

if $\bowtie$ is $\leftrightarrow$: the DAGs for $\mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)$ add 3+3 "$\wedge$"/"$\vee$" nodes and 6 + 6 arcs:

$\mathsf{NNF}(\quad \varphi_1 \leftrightarrow \varphi_2\ ) \stackrel{\text{def}}{=} (\mathsf{NNF}(\neg\varphi_1) \vee \mathsf{NNF}(\varphi_2)) \wedge (\mathsf{NNF}(\quad \varphi_1) \vee \mathsf{NNF}(\neg\varphi_2))$ and

$\mathsf{NNF}(\neg(\varphi_1 \leftrightarrow \varphi_2)) \stackrel{\text{def}}{=} (\mathsf{NNF}(\quad \varphi_1) \vee \mathsf{NNF}(\varphi_2)) \wedge (\mathsf{NNF}(\neg\varphi_1) \vee \mathsf{NNF}(\neg\varphi_2))$, thus

$$|\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle| = 18 + |\langle \mathsf{NNF}(\varphi_1), \mathsf{NNF}(\neg\varphi_1)\rangle| + |\langle \mathsf{NNF}(\varphi_2), \mathsf{NNF}(\neg\varphi_2)\rangle| \tag{14}$$

if $\varphi \stackrel{\text{def}}{=} (\varphi_1 \bowtie \varphi_2)$ s.t. $\bowtie \in \{\vee, \rightarrow\}$: these cases can be reduced to the previous cases, since

$\mathsf{NNF}(\varphi_1 \vee \varphi_2) = \mathsf{NNF}(\neg(\neg\varphi_1 \wedge \neg\varphi_2))$ and $\mathsf{NNF}(\varphi_1 \rightarrow \varphi_2) = \mathsf{NNF}(\neg(\varphi_1 \wedge \neg\varphi_2))$.

Therefore, from (12), (13) and (14) we have that $|\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi)\rangle|$ is $O(|\varphi|)$. □

A.2 Proof for Lemma 2 in Section 2.1

$\varphi_1\|_\mu$	$\top$	$\top$	$\top$	$*$	$*$	$*$	$\bot$	$\bot$	$\bot$
$\varphi_2\|_\mu$	$\top$	$*$	$\bot$	$\top$	$*$	$\bot$	$\top$	$*$	$\bot$
$\neg(\varphi_1\|_\mu)$	$\bot$	$\bot$	$\bot$	$*$	$*$	$*$	$\top$	$\top$	$\top$
$\varphi_1\|_\mu \wedge \varphi_2\|_\mu$	$\top$	$*$	$\bot$	$*$	$*$	$\bot$	$\bot$	$\bot$	$\bot$
$\varphi_1\|_\mu \vee \varphi_2\|_\mu$	$\top$	$\top$	$\top$	$\top$	$*$	$*$	$\top$	$*$	$\bot$
$\varphi_1\|_\mu \rightarrow \varphi_2\|_\mu$	$\top$	$*$	$\bot$	$\top$	$*$	$*$	$\top$	$\top$	$\top$
$\varphi_1\|_\mu \leftrightarrow \varphi_2\|_\mu$	$\top$	$*$	$\bot$	$*$	$*$	$*$	$\bot$	$*$	$\top$

Fig. 6. Three-value-semantics of $\varphi|_\mu$ in terms of $\{\top, \bot, *\}$ ("true", "false", "unknown").

In the following the symbol "$*$" denotes any formula which is not in $\{\top, \bot\}$. Following [68, 69], we adopt a 3-value semantics for residuals $\varphi|_\mu \in \{\top, \bot, *\}$, so that "$\varphi|_\mu = *$" means "$\varphi|_\mu \notin \{\top, \bot\}$" and "$\varphi_1|_\mu = \varphi_2|_\mu$" means that the two residuals $\varphi_1|_\mu$ and $\varphi_2|_\mu$ are either both $\top$, or both $\bot$, or neither is in $\{\top, \bot\}$. (Notice that, in the latter case, $\varphi_1|_\mu = \varphi_2|_\mu$ even if $\varphi_1|_\mu$ and $\varphi_2|_\mu$ are different formulas.) We extend the definition to tuples in an obvious way: $\langle \varphi_1|_\mu, \ldots, \varphi_n|_\mu \rangle = \langle \psi_1|_\mu, \ldots, \psi_n|_\mu \rangle$ iff $\varphi_i|_\mu = \psi_i|_\mu$ for each $i \in [1 \ldots n]$.

The 3-value semantics of the Boolean operators is reported for convenience in Figure 6. As a straightforward consequence of the above semantics, we have that:

$$\begin{aligned} (\neg\varphi)|_\mu &= \neg(\varphi|_\mu) \text{ (hereafter simply "}\neg\varphi|_\mu\text{")} \\ (\varphi_1 \bowtie \varphi_2)|_\mu &= \varphi_1|_\mu \bowtie \varphi_2|_\mu \text{ for } \bowtie \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}. \end{aligned}$$

Also, the usual transformations apply: $\neg(\varphi_1 \wedge \varphi_2)|_\mu = \neg\varphi_1|_\mu \vee \neg\varphi_2|_\mu$, $\neg(\varphi_1 \vee \varphi_2)|_\mu = \neg\varphi_1|_\mu \wedge \neg\varphi_2|_\mu$, $(\varphi_1 \leftrightarrow \varphi_2)|_\mu = (\neg\varphi_1|_\mu \vee \varphi_2|_\mu) \wedge (\varphi_1|_\mu \vee \neg\varphi_2|_\mu)$ and $\neg(\varphi_1 \leftrightarrow \varphi_2)|_\mu = (\varphi_1|_\mu \vee \varphi_2|_\mu) \wedge (\neg\varphi_1|_\mu \vee \neg\varphi_2|_\mu)$. For convenience, sometimes we denote as $\bar{v}$ the complement of $v \in \{\top, \bot, *\}$, i.e., $\bar{v} \stackrel{\text{def}}{=} \neg v$, so that $\bar{\top} = \bot$, $\bar{\bot} = \top$, $\bar{*} = *$.

We prove the following lemma, from which Lemma 2 in §2.1 follows directly.

Lemma 3. *Consider a formula φ, and a partial assignment μ. Then:*

$$\langle \varphi|_\mu, \neg\varphi|_\mu \rangle = \langle \mathsf{NNF}(\varphi)|_\mu, \mathsf{NNF}(\neg\varphi)|_\mu \rangle. \tag{15}$$

Proof. As in §A.1, we prove this fact by reasoning on the 2-root DAG for the pair $\langle \mathsf{NNF}(\varphi), \mathsf{NNF}(\neg\varphi) \rangle$. Specifically, we prove (15) by induction on the structure of φ.

if φ is an atom: then $\mathsf{NNF}(\varphi) = \varphi$ and $\mathsf{NNF}(\neg\varphi) = \neg\varphi$, so that $\varphi|_\mu = \mathsf{NNF}(\varphi)|_\mu$ and $\neg\varphi|_\mu = \mathsf{NNF}(\neg\varphi)|_\mu$.

if $\varphi \stackrel{\text{def}}{=} \neg\varphi_1$: we assume by induction that $\langle \varphi_1|_\mu, \neg\varphi_1|_\mu \rangle = \langle \mathsf{NNF}(\varphi_1)|_\mu, \mathsf{NNF}(\neg\varphi_1)|_\mu \rangle$. Let v be s.t. $\langle \varphi|_\mu, \neg\varphi|_\mu \rangle = \langle v, \bar{v} \rangle$. Then:

$$\begin{aligned} &\langle \neg\varphi_1|_\mu, \ \ \varphi_1|_\mu \rangle = \langle v, \bar{v} \rangle \Leftrightarrow \\ &\langle \ \ \varphi_1|_\mu, \neg\varphi_1|_\mu \rangle = \langle \bar{v}, v \rangle \stackrel{\text{ind}}{\Leftrightarrow} \\ &\langle \mathsf{NNF}(\ \ \varphi_1)|_\mu, \mathsf{NNF}(\neg\varphi_1)|_\mu \rangle = \langle \bar{v}, v \rangle \Leftrightarrow \\ &\langle \mathsf{NNF}(\neg\varphi_1)|_\mu, \mathsf{NNF}(\ \ \varphi_1)|_\mu \rangle = \langle v, \bar{v} \rangle \end{aligned}$$

if $\varphi \stackrel{\text{def}}{=} (\varphi_1 \bowtie \varphi_2)$: s.t. $\bowtie \in \{\wedge, \leftrightarrow\}$. We assume by induction that

$$\begin{aligned} &\langle \varphi_1|_\mu, \neg\varphi_1|_\mu \rangle = \langle \mathsf{NNF}(\varphi_1)|_\mu, \mathsf{NNF}(\neg\varphi_1)|_\mu \rangle, \\ &\langle \varphi_2|_\mu, \neg\varphi_2|_\mu \rangle = \langle \mathsf{NNF}(\varphi_2)|_\mu, \mathsf{NNF}(\neg\varphi_2)|_\mu \rangle. \end{aligned}$$

Then,

if $\bowtie$ is $\wedge$:

$$\begin{aligned}
&\langle (\varphi_1 \wedge \varphi_2)|_\mu, \neg(\varphi_1 \wedge \varphi_2)|_\mu \rangle = \\
&\langle \varphi_1|_\mu \wedge \varphi_2|_\mu, \neg\varphi_1|_\mu \vee \neg\varphi_2|_\mu \rangle \overset{\text{ind}}{=} \\
&\langle \mathsf{NNF}(\varphi_1)|_\mu \wedge \mathsf{NNF}(\varphi_2)|_\mu, \mathsf{NNF}(\neg\varphi_1)|_\mu \vee \mathsf{NNF}(\neg\varphi_2)|_\mu \rangle = \\
&\langle (\mathsf{NNF}(\varphi_1) \wedge \mathsf{NNF}(\varphi_2))|_\mu, (\mathsf{NNF}(\neg\varphi_1) \vee \mathsf{NNF}(\neg\varphi_2))|_\mu \rangle = \\
&\langle \mathsf{NNF}(\varphi_1 \wedge \varphi_2)|_\mu, \mathsf{NNF}(\neg(\varphi_1 \wedge \varphi_2))|_\mu \rangle
\end{aligned}$$

if $\bowtie$ is $\leftrightarrow$:

$$\begin{aligned}
&\langle (\varphi_1 \leftrightarrow \varphi_2)|_\mu, \neg(\varphi_1 \leftrightarrow \varphi_2)|_\mu \rangle = \\
&\langle ((\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2))|_\mu, ((\varphi_1 \vee \varphi_2) \wedge (\neg\varphi_1 \vee \neg\varphi_2))|_\mu \rangle = \\
&\langle (\neg\varphi_1|_\mu \vee \varphi_2|_\mu) \wedge (\varphi_1|_\mu \vee \neg\varphi_2|_\mu), (\varphi_1|_\mu \vee \varphi_2|_\mu) \wedge (\neg\varphi_1|_\mu \vee \neg\varphi_2|_\mu) \rangle \overset{\text{ind}}{=} \\
&\quad \langle (\mathsf{NNF}(\neg\varphi_1)|_\mu \vee \mathsf{NNF}(\varphi_2)|_\mu) \wedge (\mathsf{NNF}(\varphi_1)|_\mu \vee \mathsf{NNF}(\neg\varphi_2)|_\mu), \\
&\quad (\mathsf{NNF}(\varphi_1)|_\mu \vee \mathsf{NNF}(\varphi_2)|_\mu) \wedge (\mathsf{NNF}(\neg\varphi_1)|_\mu \vee \mathsf{NNF}(\neg\varphi_2)|_\mu) \rangle = \\
&\quad \langle ((\mathsf{NNF}(\neg\varphi_1) \vee \mathsf{NNF}(\varphi_2)) \wedge (\mathsf{NNF}(\varphi_1) \vee \mathsf{NNF}(\neg\varphi_2)))|_\mu, \\
&\quad ((\mathsf{NNF}(\varphi_1) \vee \mathsf{NNF}(\varphi_2)) \wedge (\mathsf{NNF}(\neg\varphi_1) \vee \mathsf{NNF}(\neg\varphi_2)))|_\mu \rangle = \\
&\quad \langle \mathsf{NNF}(\varphi_1 \leftrightarrow \varphi_2)|_\mu, \mathsf{NNF}(\neg(\varphi_1 \leftrightarrow \varphi_2))|_\mu \rangle.
\end{aligned}$$

if $\varphi \overset{\text{def}}{=} (\varphi_1 \bowtie \varphi_2)$: s.t. $\bowtie \in \{\vee, \rightarrow\}$. These cases can be reduced to the previous cases, since $\mathsf{NNF}(\varphi_1 \vee \varphi_2) = \mathsf{NNF}(\neg(\neg\varphi_1 \wedge \neg\varphi_2))$ and $\mathsf{NNF}(\varphi_1 \rightarrow \varphi_2) = \mathsf{NNF}(\neg(\varphi_1 \wedge \neg\varphi_2))$.

□

Thus, $\langle \varphi|_\mu, \neg\varphi|_\mu \rangle = \langle \mathsf{NNF}(\varphi)|_\mu, \mathsf{NNF}(\neg\varphi)|_\mu \rangle$ so that $\varphi|_\mu = v$ iff $\mathsf{NNF}(\varphi)|_\mu = v$ for every $v \in \{\top, \bot\}$, so that Lemma 2 in §2.1 holds.

B An Analysis of Candidate Solvers

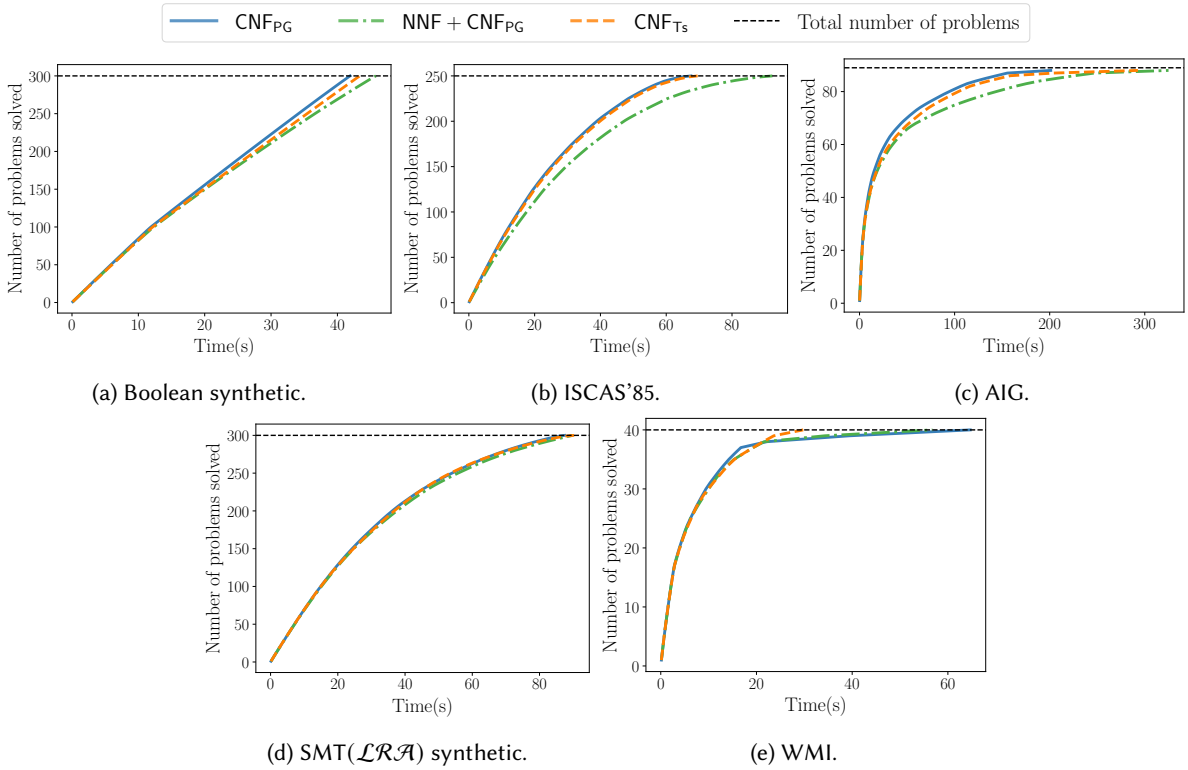
In Table 1 we report an analysis of the features of every candidate AllSAT and AllSMT solver, as discussed in §5.1.

		Required features					Options		
	Solver	Available	CNF input	Projected	Partial	Minimal	Disjoint	Non-disjoint	Notes
AllSAT	RELSAT	✓	✓	✗	✗	✗	✓	✗	
	Grumberg	✗	✓	✓	✓	✓	✓	✓	Ex-post minimization
	SOLALL	?	✓	✓	✓	✗	✓	✗	Result stored in (O)BDD
	Jin	✗	✗	✗	✓	✓	✗	✓	AIG input
	clasp	✓	✓	✓	✗	✗	✓	✗	ASP solver
	PicoSAT	✓	✓	✗	✗	✗	✓	✗	
	Yu	?	✓	✗	✓	✓	✓	✓	
	BC	✓	✓	✗	✓	✓	✓	✓	
	NBC	✓	✓	✗	✗	✗	✓	✗	
	BDD	✓	✓	✗	✓	✗	✗	✓	Result stored in (O)BDD
	depbdd	✗	✓	✓	✓	✗	✓	✗	Result stored in (O)BDD
	Dualiza	✓	✓	✓	✓	✗	✓	✓	Dual-reasoning-based
	BASolver	✗	✓	✗	✓	✓	✗	✓	
	AllSATCC	✗	✓	✗	✓	✓	✓	✗	
	HALL	✓	✗	✗	✓	✓	✓	✓	AIG input
	TabularAllSAT	✓	✓	✓	✓	✗	✓	✗	
	model-graph	✓	✗	✗	✓	✗	✓	✗	d-DNNF input
AllSMT	MathSAT	✓	✓	✓	✓	✓	✓	✓	
	aZ3	✓	✓	✓	✗	✗	✓	✗	
	TabularAllSMT	✓	✓	✓	✓	✗	✓	✗	

Table 1. List of AllSAT and AllSMT solvers (rows) and supported features (columns). (In the "Available" column, "✗" indicates that the authors confirmed to us the unavailability of their tool, whereas "?" indicates that they did not reply to our enquiry.)

C Experimental Results on Plain SAT and SMT Solving

In this section, we report the results of the experiments on plain SAT and SMT solving with MathSAT using the different CNF-izations. The plots in Figure 7 show that NNF+CNF_{PG} does not bring any advantage for plain SAT solving, supporting the analysis in §5.5.



(a) Boolean synthetic. (b) ISCAS'85. (c) AIG.

(d) SMT($\mathcal{LRA}$) synthetic. (e) WMI.

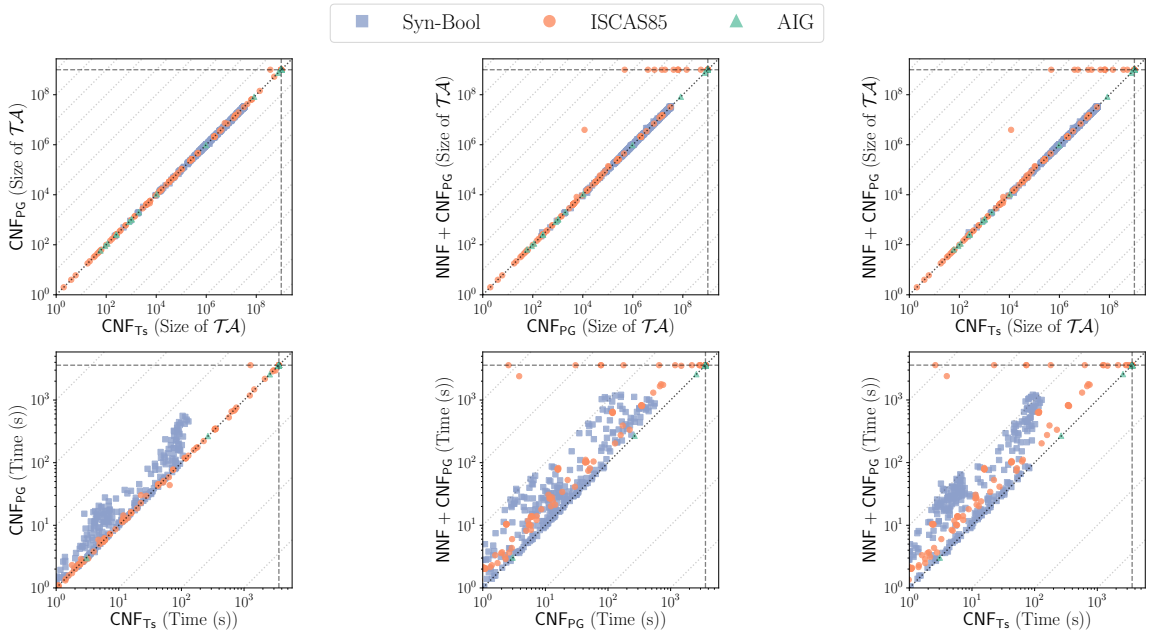
Fig. 7. Time taken for plain SAT and SMT solving using the different CNF transformations. The y-axis reports the number of instances for which the solver finished within the cumulative time on the x-axis.

D Experimental Results on d-DNNF-based Enumeration

Recently, in [37], the authors presented a tool, named MODEL-GRAPH, to enumerate all models of a d-DNNF formula [18].

Following the D4+MODEL-GRAPH approach [37], we first compile the CNF-ization of the input formula into a d-DNNF using the D4 compiler [38], and then enumerate its models using MODEL-GRAPH. To avoid enumerating on CNF labels, we project the d-DNNF onto the original atoms $\mathbf{A}$ of the input formula, and then use MODEL-GRAPH to enumerate the models of the projected d-DNNF.

The results of the experiments on the Boolean benchmarks are shown in Figure 8. We see that $\mathsf{CNF}_{\mathsf{Ts}}$ is uniformly the best-performing CNF-ization, supporting the analysis in §5.5



(a) Results for disjoint enumeration.

Bench.	Instances	T.O. for disjoint AllSAT		
		$\mathsf{CNF}_{\mathsf{Ts}}$	$\mathsf{CNF}_{\mathsf{PG}}$	$\mathsf{NNF+CNF}_{\mathsf{PG}}$
Syn-Bool	300	**0**	**0**	**0**
ISCAS85	250	**15**	16	27
AIG	89	**76**	**76**	**76**

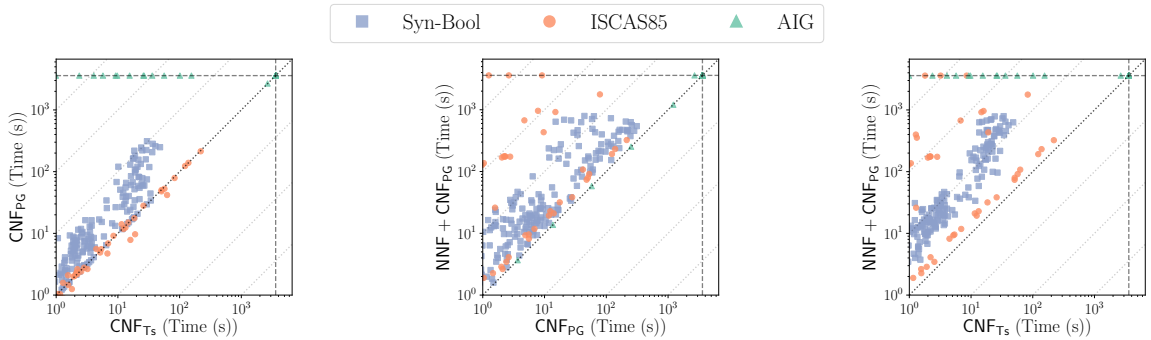
(b) Number of timeouts.

Fig. 8. Results on the Boolean benchmarks using D4+MODEL-GRAPH. Plots in 8a compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 8b. All axes use a logarithmic scale.

E Experimental Results on d-DNNF-based Model Counting

We tested the different CNF-izations also for model counting using D4. Since CNF_{PG} and $\text{NNF+CNF}_{\text{PG}}$ do not preserve the model count, we perform projected model counting on the original atoms of the input formula. Even though this would not be necessary for CNF_{Ts}, we apply the same procedure to ensure a fair comparison.

The results of the experiments on the Boolean benchmarks are shown in Figure 9. We can observe that also for model counting, CNF_{Ts} is uniformly the best-performing CNF-ization, supporting the analysis in §5.5.



(a) Results for model counting.

Bench.	Instances	T.O. for disjoint AllSAT		
		CNF_{Ts}	CNF_{PG}	$\text{NNF+CNF}_{\text{PG}}$
Syn-Bool	300	**0**	**0**	**0**
ISCAS85	250	**0**	**0**	4
AIG	89	**50**	67	68

(b) Number of timeouts.

Fig. 9. Results on the Boolean benchmarks using D4 for model counting. Plots in 9a compare CNF-izations by execution time. Points on dashed lines represent timeouts, shown in 9b. All axes use a logarithmic scale.

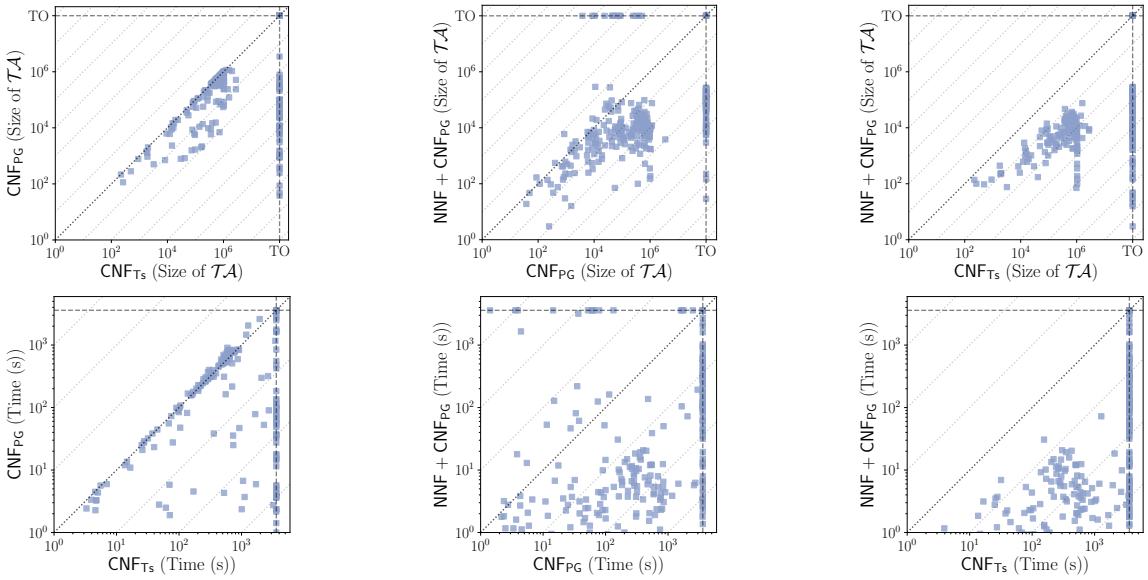
F Details on Experimental Results in the Paper

In this section, we report the scatter plots on individual benchmarks for the experiments presented in §5.4, Figures 1, 2 and 3.

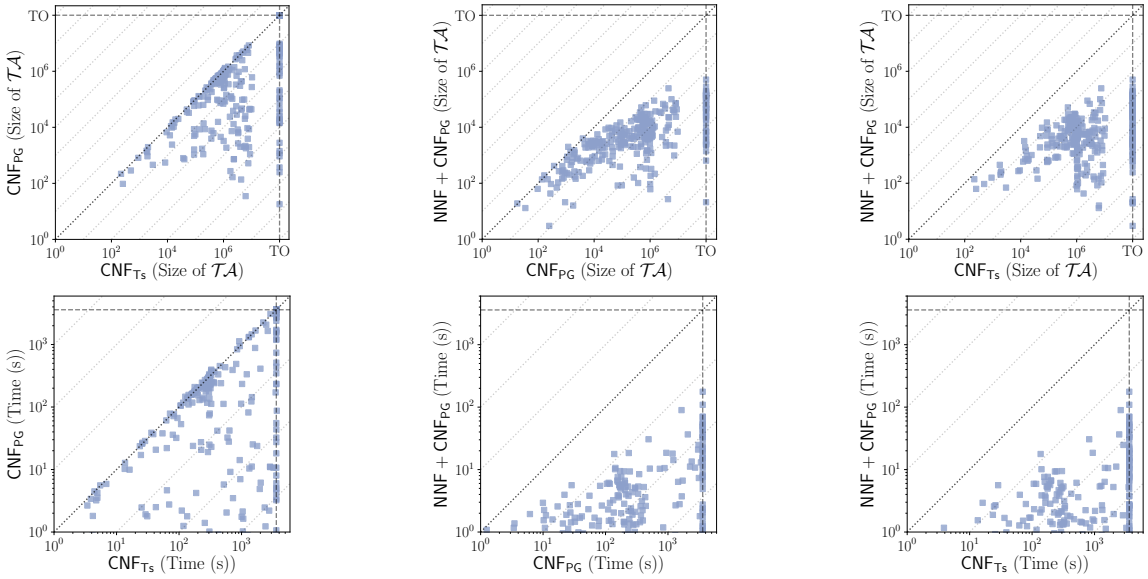
For AllSAT, Figures 10, 11 and 12 show the results for MathSAT on the Boolean synthetic, ISCAS'85 and AIG benchmarks, respectively. Figures 13, 14 and 15 show the results for TabularAllSAT on the same benchmarks.

For AllSMT, Figures 16 and 17 show the results for MathSAT on the SMT($\mathcal{LRA}$) synthetic and WMI benchmarks, respectively. Figures 18 and 19 show the results for TabularAllSMT on the same benchmarks.

Received 9 February 2024; revised 19 November 2024; accepted 30 April 2025

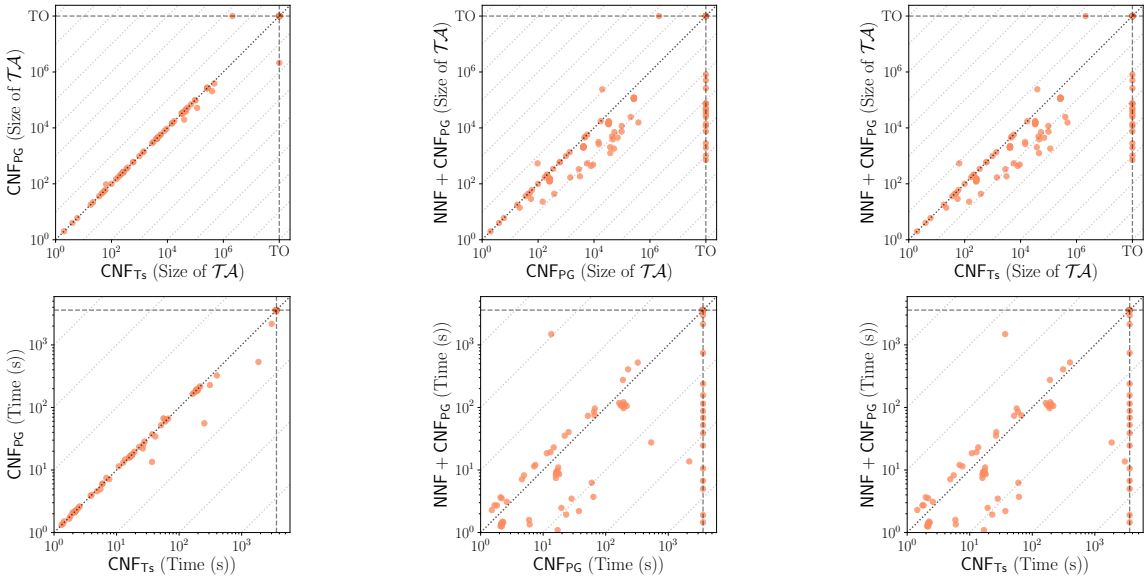


(a) Results for disjoint enumeration.

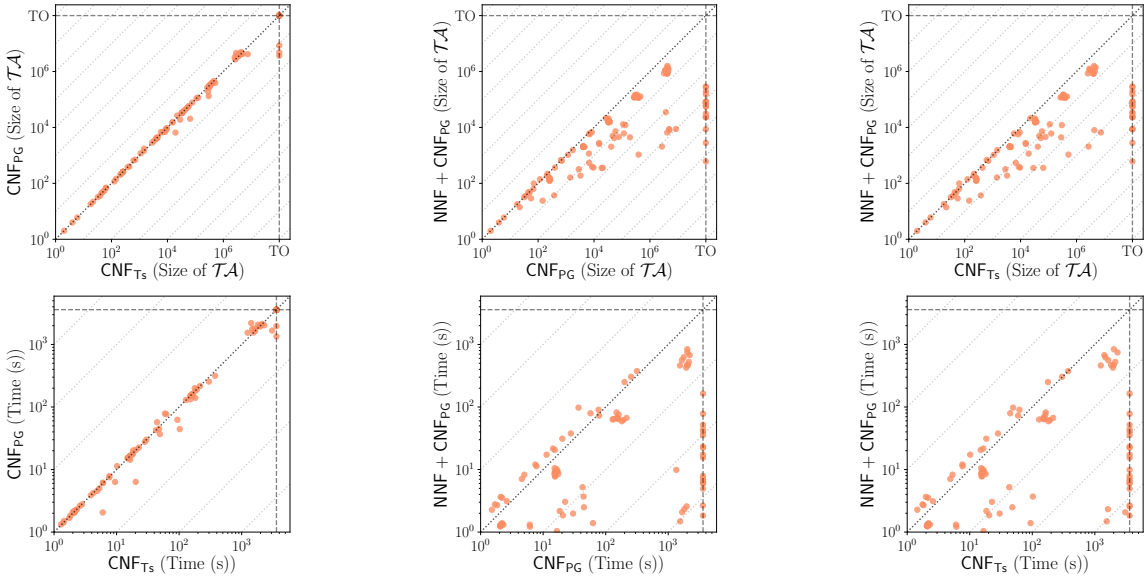


(b) Results for non-disjoint enumeration.

Fig. 10. Results on the Boolean synthetic benchmarks using MathSAT. Plots in 10a and 10b compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 1c. All axes use a logarithmic scale.

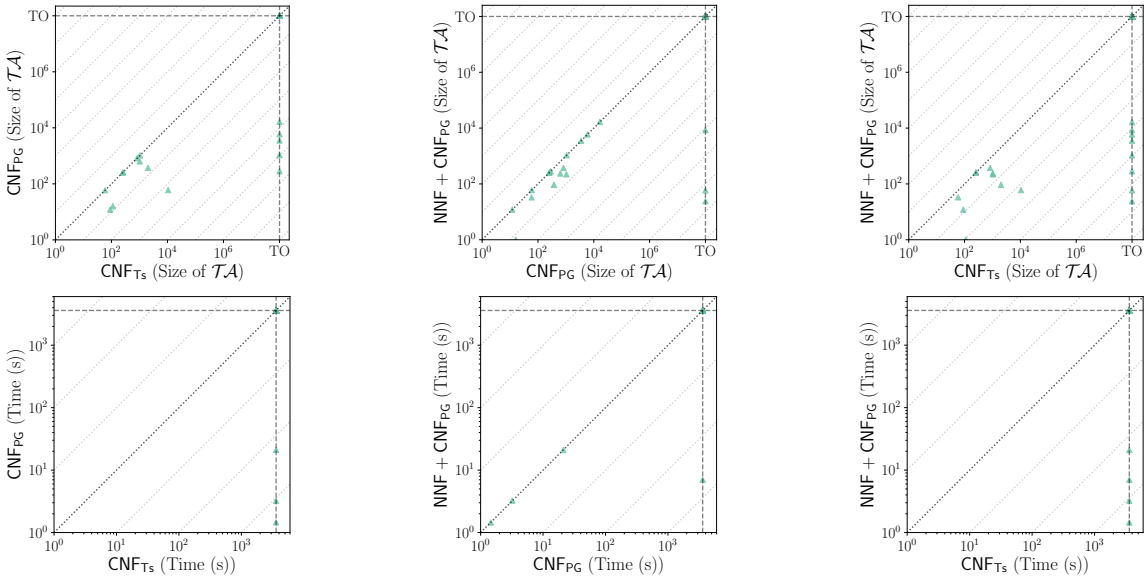


(a) Results for disjoint enumeration.

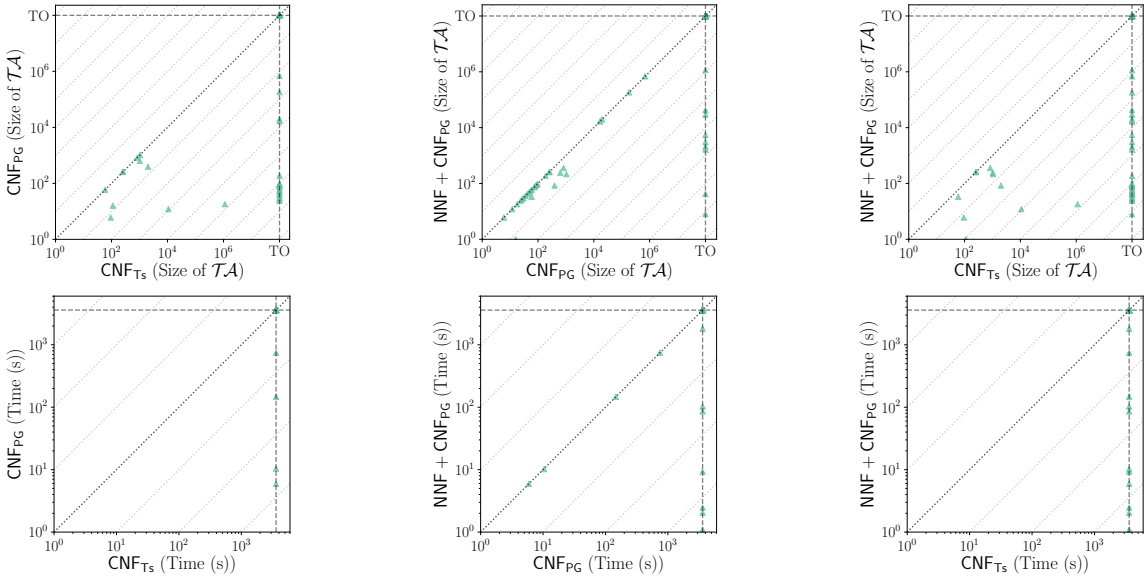


(b) Results for non-disjoint enumeration.

Fig. 11. Results on the ISCAS'85 benchmarks using MathSAT. Plots in 11a and 11b compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 1c. All axes use a logarithmic scale.

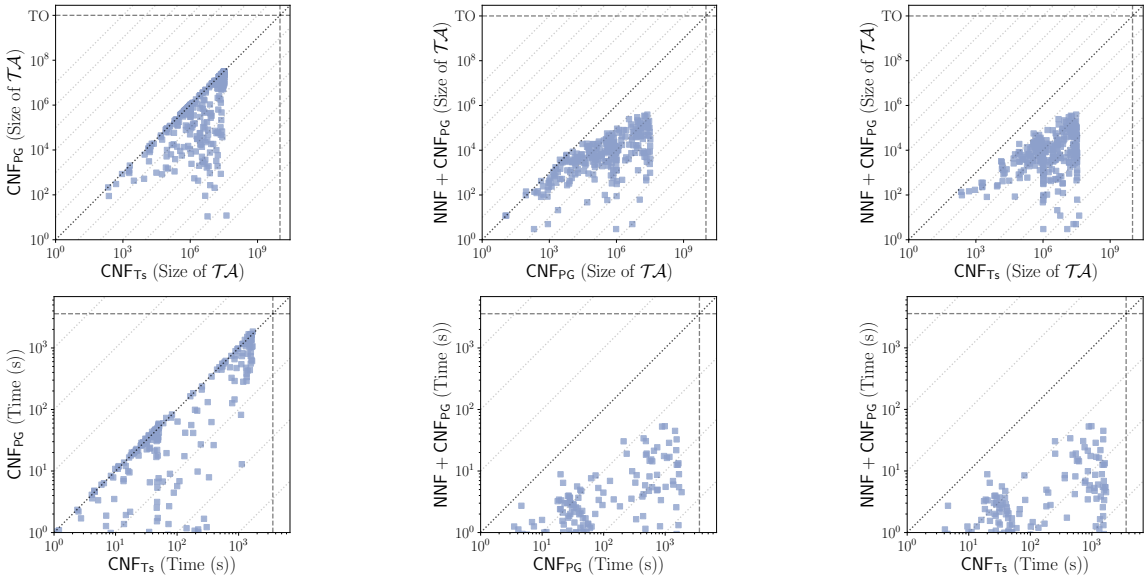


(a) Results for disjoint enumeration.



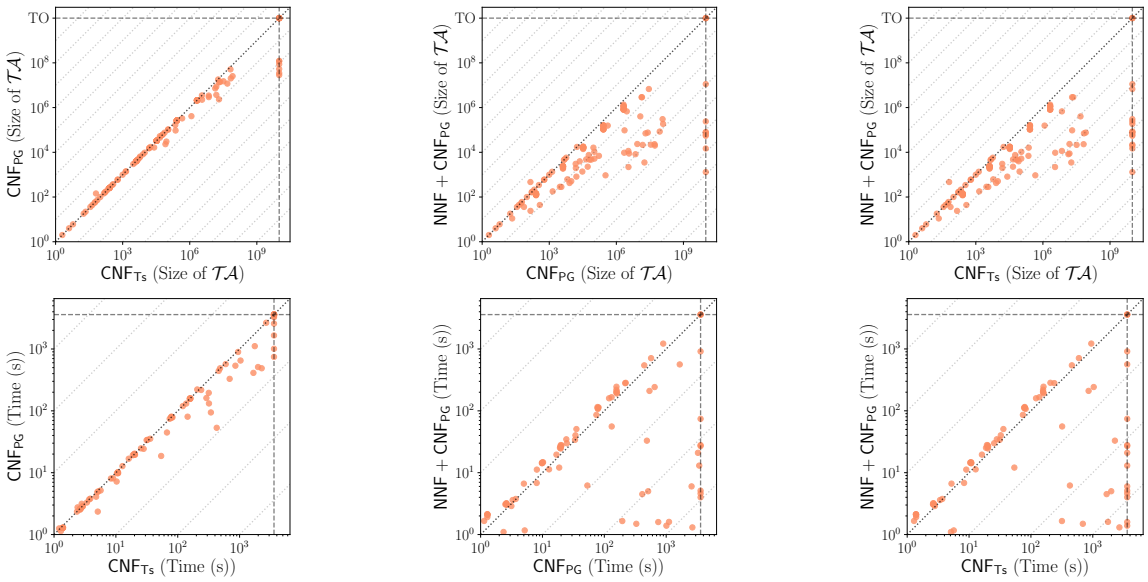
(b) Results for non-disjoint enumeration.

Fig. 12. Results on the AIG benchmarks using MathSAT. Plots in 12a and 12b compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 1c. All axes use a logarithmic scale.



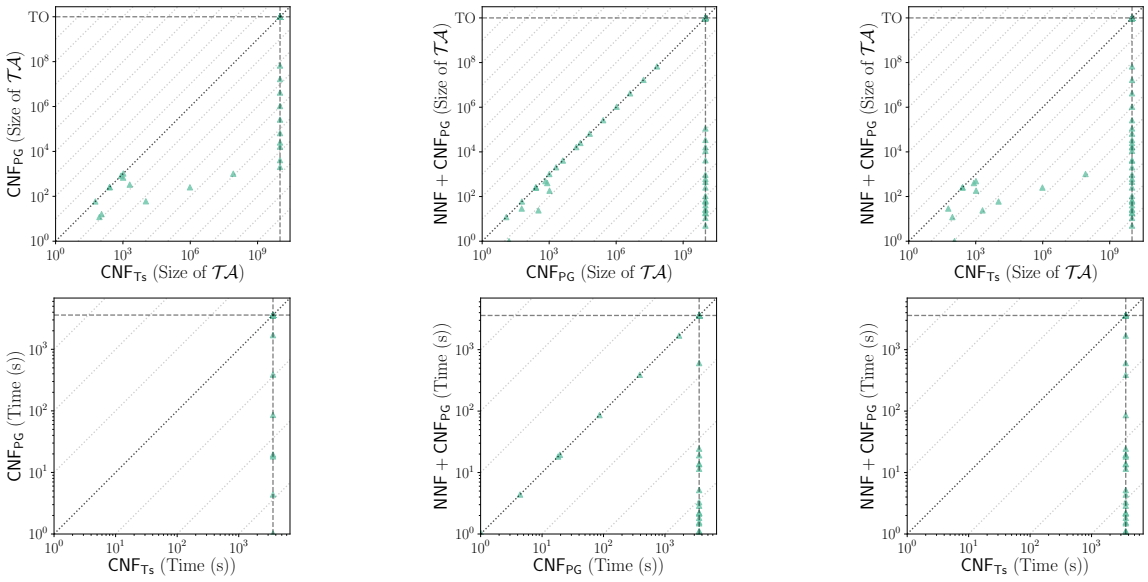
(a) Results for disjoint enumeration.

Fig. 13. Results on the Boolean synthetic benchmarks using TabularAllSAT. Plots in 13a compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 2b. All axes use a logarithmic scale.



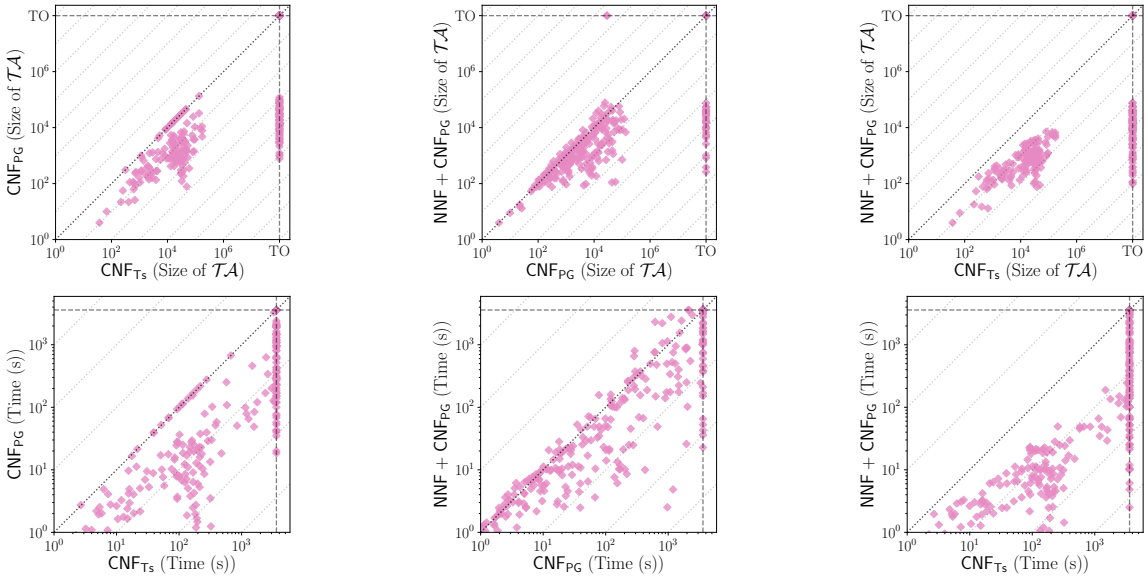
(a) Results for disjoint enumeration.

Fig. 14. Results on the ISCAS'85 benchmarks using TabularAllSAT. Plots in 14a compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 2b. All axes use a logarithmic scale.

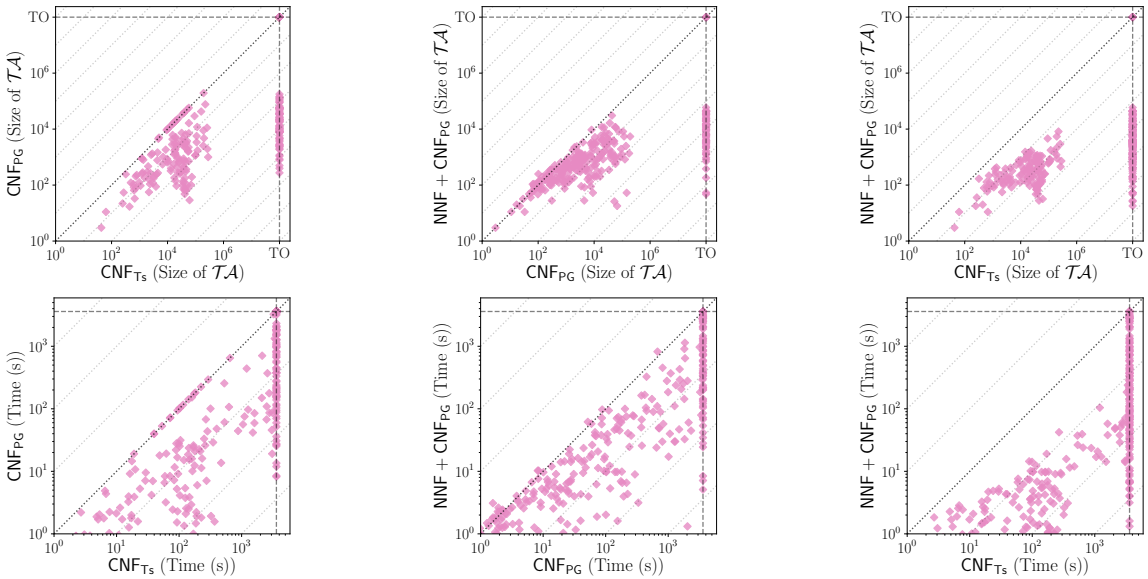


(a) Results for disjoint enumeration.

Fig. 15. Results on the AIG benchmarks using TabularAllSAT. Plots in 15a compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 2b. All axes use a logarithmic scale.

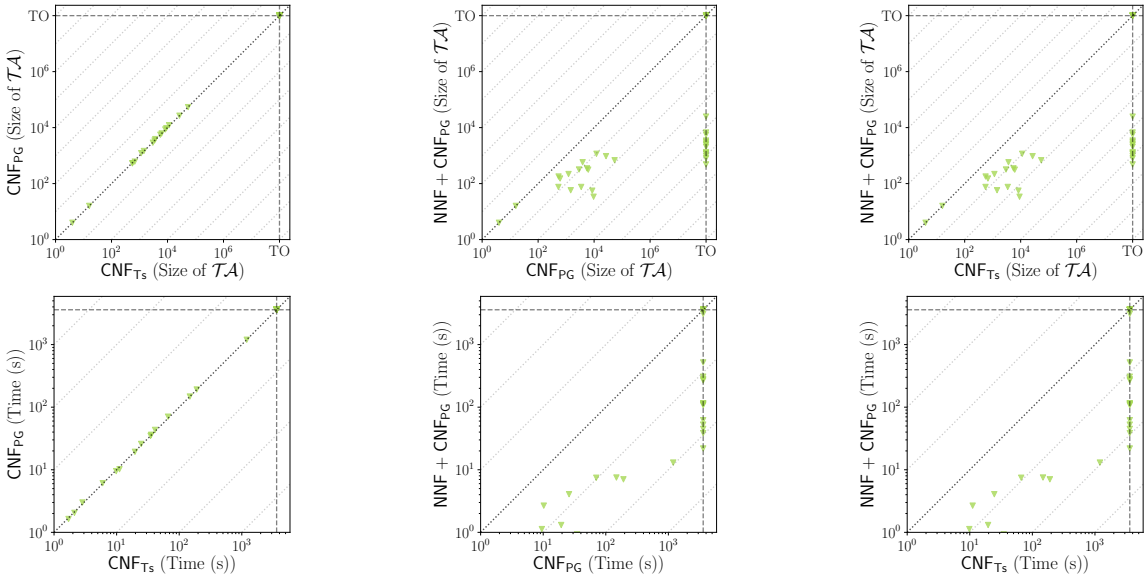


(a) Results for disjoint enumeration.

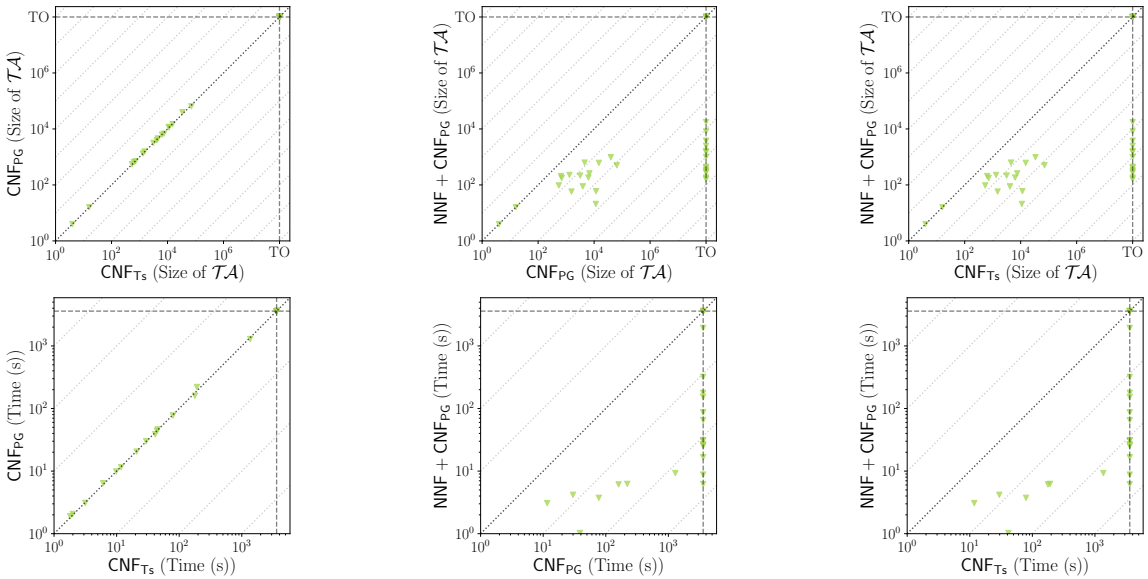


(b) Results for non-disjoint enumeration.

Fig. 16. Results on the SMT($\mathcal{LRA}$) synthetic benchmarks using MathSAT. Plots in 16a and 16b compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 3c. All axes use a logarithmic scale.

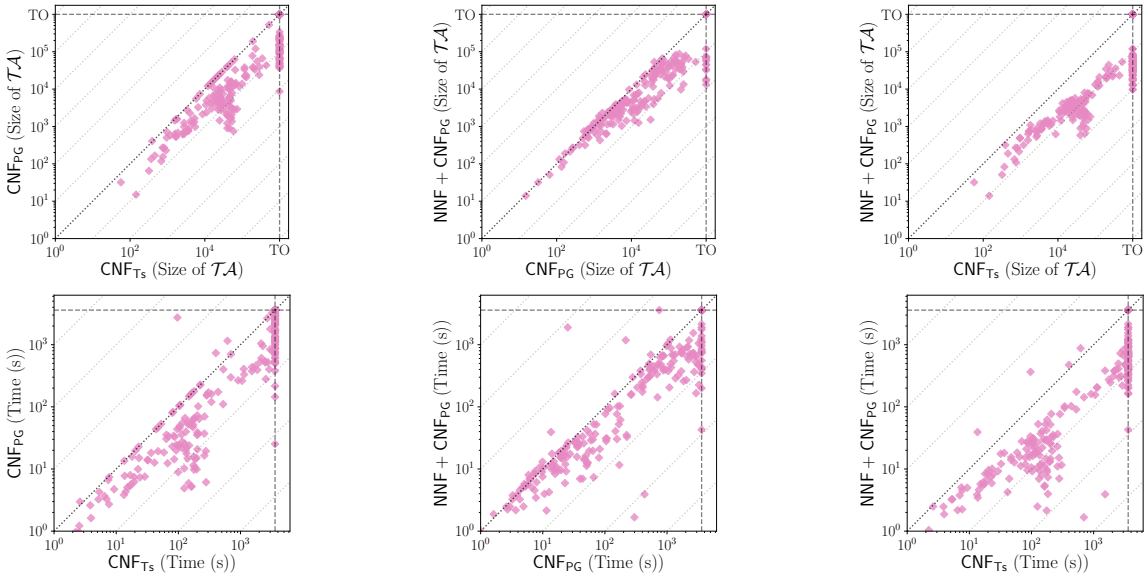


(a) Results for disjoint enumeration.



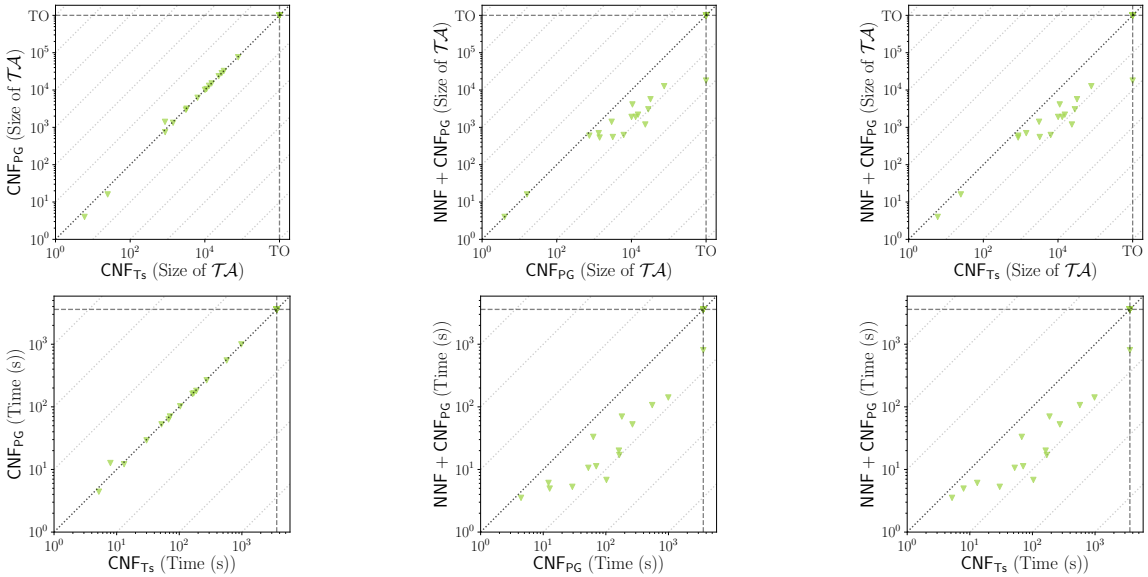
(b) Results for non-disjoint enumeration.

Fig. 17. Results on the WMI benchmarks using MathSAT. Plots in 17a and 17b compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 3c. All axes use a logarithmic scale.



(a) Results for disjoint enumeration.

Fig. 18. Results on the SMT($\mathcal{LRA}$) synthetic benchmarks using TabularAllSMT. Plots in 18a compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 4b. All axes use a logarithmic scale.



(a) Results for disjoint enumeration.

Fig. 19. Results on the WMI benchmarks using TabularAllSMT. Plots in 19a compare CNF-izations by $\mathcal{TA}$ size (first row) and execution time (second row). Points on dashed lines represent timeouts, shown in 4b. All axes use a logarithmic scale.