Definability for model counting

Jean-Marie Lagniez, Emmanuel Lonca, Pierre Marquis

▶ **To cite this version:**

Jean-Marie Lagniez, Emmanuel Lonca, Pierre Marquis. Definability for model counting. Artificial Intelligence, 2020, 281, pp.103229. 10.1016/j.artint.2019.103229 . hal-03167468

HAL Id: hal-03167468
https://univ-artois.hal.science/hal-03167468v1

Submitted on 7 Mar 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial 4.0 International License

Version of Record: https://www.sciencedirect.com/science/article/pii/S0004370218303928
Manuscript_b412b94c040a2e7ada4263b92d1326a0

Definability for Model Counting☆

Jean-Marie Lagniez[a], Emmanuel Lonca[a], Pierre Marquis[b]

[a]*CRIL, CNRS UMR8188 - U. Artois,*
rue Jean Souvraz – F-62307 Lens France

[b]*CRIL, CNRS UMR8188 - U. Artois & Institut Universitaire de France,*
rue Jean Souvraz – F-62307 Lens France

Abstract

We define and evaluate a new preprocessing technique for propositional model counting. This technique leverages definability, i.e., the ability to determine that some gates are implied by the input formula Σ. Such gates can be exploited to simplify Σ without modifying its number of models. Unlike previous techniques based on gate detection and replacement, gates do not need to be made explicit in our approach. Our preprocessing technique thus consists of two phases: computing a bipartition $\langle \mathrm{I}, \mathrm{O} \rangle$ of the variables of Σ where the variables from O are defined in Σ in terms of I, then eliminating some variables of O in Σ. Our experiments show the computational benefits which can be achieved by taking advantage of our preprocessing technique for model counting.

Keywords: definability, model counting.

1. Introduction

Propositional model counting (also known as the #SAT problem) is the task of computing the number of models of a given propositional formula Σ. This prob-

☆This is an extended version of the paper entitled "Improving Model Counting by Leveraging Definability" published in the proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16), New York City, 2016 (751-757).

Email addresses: lagniez@cril.univ-artois.fr (Jean-Marie Lagniez), lonca@cril.univ-artois.fr (Emmanuel Lonca), marquis@cril.univ-artois.fr (Pierre Marquis)

© 2020 published by Elsevier. This manuscript is made available under the CC BY NC user license
https://creativecommons.org/licenses/by-nc/4.0/

lem and its direct generalization, weighted model counting,[1] are central to many AI problems including probabilistic inference [1, 2, 3, 4] and forms of planning [5, 6]. They have also many applications outside AI, like in SAT-based automatic test pattern generation, for evaluating the vulnerability to malicious fault attacks in hardware circuits (see e.g., [7]).

However, propositional model counting (as well as WMC, which can be reduced to #SAT [8]) are computationally hard (they are #P-complete problems [9, 10, 11]), actually much harder in practice than the satisfiability problem (SAT). The power offered by the ability to count efficiently and, dually, the difficulty to do it, are well-reflected by Toda's theorem, showing that $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$, i.e., every problem from the polynomial hierarchy PH can be solved in polynomial time provided that a #P oracle is available [12]. Nevertheless, the significance of #SAT explains why much effort has been spent for the last decade in developing new algorithms for model counting (either exact or approximate), which prove practical for larger and larger instances, see e.g., [13, 14, 15, 16, 17, 18].

In this paper, we present a new preprocessing technique for improving exact model counting from the computational side. Preprocessing techniques are nowadays acknowledged as computationally valuable for a number of automated reasoning tasks, especially SAT solving and QBF solving [19, 20, 21, 22, 23, 24, 25, 26, 27]. As such, they are now embodied in state-of-the-art SAT solvers, like `Glucose` [28] which takes advantage of the `SatELite` preprocessor [22], `Lingeling` [29] which has an internal preprocessor, and `Riss` [30] which takes advantage of the `Coprocessor` preprocessor [31].

Our approach elaborates on previous preprocessing techniques [32] that can be exploited for improving the model counting task from a computational standpoint. Among them is *gate detection and replacement*. Basically, every variable y of the input formula Σ that turns out to be defined in Σ in terms of other variables $X = \{x_1, \ldots, x_k\}$ can be replaced by the corresponding gate Φ_X, while preserving the number of models of Σ. Indeed, whenever a partial assignment over the variables of X is considered, either it is jointly inconsistent with Σ or in every model of Σ that extends this partial assignment, y has the same truth value. Literal equivalences, AND/OR gates and XOR gates can be detected (ei-

[1] In weighted model counting (WMC), each literal is associated with a real number, the weight of an interpretation is the product of the weights of the literals it sets to true, and the weight of a formula is the sum of the weights of its models. Accordingly, WMC amounts to model counting when each literal has weight 1.

ther syntactically or using Boolean constraint propagation). The corresponding preprocessing techniques are integrated into a preprocessor, called pmc. The empirical results reported in [32] clearly show that huge computational benefits can be achieved through the detection and the replacement of gates. However, pmc remains limited due to the small number of families of gates which are targeted (literal, AND, XOR gates and their negations).

In order to fill the gap, we present in this paper a preprocessing technique to model counting that exploits in a much more aggressive way the gates that can be found in the input formula Σ. The key idea underlying this preprocessing technique is that *one does not need to identify the gates themselves but determining that they exist is enough*. To be more precise, it proves sufficient to detect that some definability relations between variables hold, without needing to identify the corresponding gates. This distinction is of tremendous importance for two reasons. On the one hand, the search space for the possible gates Φ_X is very large: it contains 2^{2^k} elements up to logical equivalence, when X contains k variables. On the other hand, in the general case, the size of any explicit gate Φ_X of y in Σ is not polynomially bounded in $|\Sigma| + |X|$ unless $\mathsf{NP} \cap \mathsf{coNP} \subseteq \mathsf{P/poly}$ (which is considered unlikely in complexity theory) [33].

What this paper mainly gives is the description and the evaluation of a new preprocessor for model counting, called B + E. The preprocessor B + E associates with a given CNF formula Σ a CNF formula Φ which has the same number of models as Σ, but is at least as simple as Σ with respect to the number of variables and the number of clauses. Requiring the input Σ to be a CNF formula is not a major restriction since any propositional circuit Σ can be turned in time linear in its size into a CNF formula having the same number of models, thanks to Tseitin's transformation [34]. Indeed, this transformation consists in adding gates to the input so that the new variables which are introduced are defined from the original ones. By the way, it is worth noting that other CNF translations, like Plaisted/Greenbaum's one [35], cannot be used. Indeed, Plaisted/Greenbaum's transformation preserves the satisfiability of the input circuit (and actually a bit more, namely the set of logical consequences of Σ over its set of variables) but not its number of models.

Example 1. *For instance, using Tseitin's transformation, the input* DNF *formula* $\Sigma = (a \wedge b) \vee (b \wedge c)$ *is associated with the* CNF *formula* $s_0 \wedge (\neg s_0 \vee s_1 \vee s_2) \wedge (s_0 \vee \neg s_1) \wedge (s_0 \vee \neg s_2) \wedge (\neg s_1 \vee a) \wedge (\neg s_1 \vee b) \wedge (s_1 \vee \neg a \vee \neg b) \wedge (\neg s_2 \vee b) \wedge (\neg s_1 \vee c) \wedge (s_2 \vee \neg b \vee \neg c)$. *This* CNF *formula is equivalent to* $s_0 \wedge [s_0 \leftrightarrow (s_1 \vee s_2)] \wedge [s_1 \leftrightarrow (a \wedge b)] \wedge [s_2 \leftrightarrow (b \wedge c)]$, *and it has precisely the same number of models*

over its set of variables as Σ *over* $\{a, b, c\}$*, viz* 3. *Using Plaisted/Greenbaum's transformation,* Σ *is associated with the* CNF *formula* $s_0 \wedge (\neg s_0 \vee s_1 \vee s_2) \wedge (\neg s_1 \vee a) \wedge (\neg s_1 \vee b) \wedge (\neg s_2 \vee b) \wedge (\neg s_1 \vee c)$*. This* CNF *formula is equivalent to* $s_0 \wedge [s_0 \rightarrow (s_1 \vee s_2)] \wedge [s_1 \rightarrow (a \wedge b)] \wedge [s_2 \rightarrow (b \wedge c)]$*, which has* 5 *models (and not* 3*) over its set of variables.*

Interestingly, the CNF format is the one considered by state-of-the-art model counters. As its name suggests it, B + E consists of two parts: B which aims to determine a *Bipartition* $\langle I, O \rangle$ of the variables of Σ such that every variable of O is defined in Σ in terms of the remaining variables (in I), and E which aims to *Eliminate* in Σ some variables of O.

More in detail, the contribution of the paper consists of the presentation of the algorithms B and E, a property establishing the correctness of the preprocessing technique used, and some empirical evidence showing the computational improvements achieved by B + E compared to the case when no preprocessor is used upstream, and also to the case when pmc is used. This paper significantly extends the results presented in [36], by providing several new contributions. Some of them are theory-oriented, including the identification of the complexity of deciding whether a definability bipartition is a subset-minimal one, and the proof that the bipartition component of our preprocessor actually computes a subset-minimal definability bipartition. Other contributions consist of empirical results; thus, some empirical evidence related to the use of d4 [37] as a downstream model counter has been added. Compared to [36], we also consider the use of B + E for approximate compilation with controllable variables, report some additional experimental results concerning classical planning benchmarks, and discuss the connections between approximate compilation with controllable variables and the projected model counting task [38, 39]. In addition, we show our approach as closely related to the notion of independent support used for approximate model counting. Especially, we compare the bipartitioner B used in our approach with the one (called MIS) used in [40].

The benchmarks used, the implementation (binary code) of B + E, and detailed empirical results are available online on www.cril.fr/KC/.

The rest of the paper is organized as follows. Section 2 gives some background on propositional definability. In Section 3, we introduce our preprocessor B + E and prove that the preprocessing technique it implements is correct. Section 4 presents results from our large scale experiments, showing B + E as a competitive preprocessor for model counting, especially when compared with pmc. Some other related work is discussed in Section 6. Finally, Section 7 concludes the

paper and lists some perspectives for further research.

2. Propositional Logic and Definability

The formal setting of our study is classical propositional logic (see e.g., [41]). Let $\mathcal{L}_{\mathcal{P}}$ be the propositional language defined inductively from a non-empty, finite set $\mathcal{P}$ of propositional variables, the usual connectives ($\neg$, $\vee$, $\wedge$, $\leftrightarrow$, etc.) and including the Boolean constants $\top$ and $\bot$. Formulae are interpreted in the classical way. The cardinality of a set S of formulae is denoted by #(S). An *interpretation* ω is a mapping from $\mathcal{P}$ to $\{0,1\}$. An interpretation ω is a *model* (resp. a *countermodel*) of a formula $\Sigma \in \mathcal{L}_{\mathcal{P}}$ if and only if ω makes Σ true (resp. false) in the usual truth functional way. Whenever a formula has a model, it is said to be *consistent* (or *satisfiable*). In the remaining case, it is *inconsistent* (or *unsatisfiable*). When a formula has no countermodel, it is a *valid* formula. For any pair of formulae Σ_1 and Σ_2, Σ_2 is a logical consequence of Σ_1, noted $\Sigma_1 \models \Sigma_2$, whenever every model of Σ_1 is a model of Σ_2; Σ_1 and Σ_2 are logically equivalent, noted $\Sigma_1 \equiv \Sigma_2$, whenever they have the same models. For any formula Σ from $\mathcal{L}_{\mathcal{P}}$, $Var(\Sigma)$ is the set of variables from $\mathcal{P}$ occurring in Σ, and $\|\Sigma\|$ is the number of models of Σ over $Var(\Sigma)$.

Among the formulae are the literals, the terms, the clauses and the CNF formulae. A *literal* ℓ is a propositional variable x (in this case, ℓ is a *positive literal*), or a negated variable $\neg$x (in this case, ℓ is a *negative literal*). When ℓ is a literal over x, i.e., ℓ = x or $\ell = \neg$x, its *complementary literal* $\sim\ell$ is given by $\sim\ell = \neg$x if ℓ = x and $\sim\ell$ = x if $\ell = \neg$x. $var(\ell)$ denotes the variable upon which ℓ is built, i.e., var(x) = $var(\neg$x) = x. A *term* is a conjunction of literals or $\top$, and a *clause* is a disjunction of literals or $\bot$. Terms and clauses are also viewed as sets of literals when it is convenient (the empty set corresponds to $\top$ when the set of literals must be interpreted as a term – the empty conjunction of literals – and to $\bot$ when it must be interpreted as a clause – the empty disjunction of literals). $\top$ is the sole valid term, and $\bot$ is the sole inconsistent clause. A *canonical term* γ_{X} over X is a consistent term in which every variable from X appears (either as a positive literal or as a negative one, i.e., as a negated variable). A CNF formula is a conjunction of clauses, also viewed as a set of clauses when this is convenient.

Let X be any finite subset of $\mathcal{P}$. The *conditioning* of a formula Σ by a consistent term γ is the formula $\Sigma \mid \gamma$ obtained by substituting in Σ for every literal ℓ of γ such that $var(\ell)$ = x every occurrence of x by $\bot$ (resp. $\top$) if ℓ is a negative (resp. a positive) literal.

Example 2. *Suppose that* Σ *is the* CNF *formula* $(a \vee d) \wedge (a \vee b) \wedge (\neg b \vee c)$ *and that* $\gamma = \neg a \wedge b$. *Then* $\Sigma \mid \gamma$ *is the formula* $(\bot \vee d) \wedge (\bot \vee \top) \wedge (\neg \top \vee c)$. *This formula can be simplified in polynomial time into the equivalent* CNF *formula* $d \wedge c$ *using simple laws of Boolean calculus.*

A formula φ is *independent* of a set X of variables if and only if φ is equivalent to a formula ψ such that $\mathit{Var}(\psi) \cap X = \emptyset$. $\exists X.\Sigma$ denotes a formula from $\mathcal{L}_{\mathcal{P}}$ equivalent to the *forgetting* of X in Σ, i.e., the strongest logical consequence of Σ (up to logical equivalence), that is independent of X [42]. "Logically strongest" means that for every formula φ that is independent of X and such that $\Sigma \models \varphi$, we have $\exists X.\Sigma \models \varphi$. The formula $\exists X.\Sigma$ can be defined inductively as follows:

- $\exists \emptyset.\Sigma = \Sigma$,
- $\exists \{x\}.\Sigma = (\Sigma \mid \neg x) \vee (\Sigma \mid x)$,
- $\exists X' \cup \{x\}.\Sigma = \exists X'.(\exists \{x\}.\Sigma)$.

When $\Sigma = \bigwedge_{i=1}^{k} \delta_i$ is a CNF formula and $X = \{x\} \cup X'$, a formula equivalent to $\exists X.\Sigma$ can be computed in a recursive way by eliminating x in Σ, obtaining thus a new CNF formula equivalent to $\exists \{x\}.\Sigma$, in which the variables of X' are then eliminated. Eliminating x in Σ basically amounts to applying the resolution principle: $\exists \{x\}.\Sigma$ is equivalent to the CNF formula consisting of the clauses δ_i of Σ such that $\mathit{Var}(\delta_i) \cap X = \emptyset$ conjoined with all the resolvents on x of the clauses of Σ.

Example 3. *As a matter of illustration, suppose that* Σ *is the* CNF *formula* $(a \vee d) \wedge (a \vee b) \wedge (\neg b \vee c)$ *and that* $X = \{b\}$. *Then* $\exists X.\Sigma$ *is the formula* $((a \vee d) \wedge (a \vee \bot) \wedge (\neg \bot \vee c)) \vee ((a \vee d) \wedge (a \vee \top) \wedge (\neg \top \vee c))$. *This formula is equivalent to the* CNF *formula* $(a \vee d) \wedge (a \vee c)$ *obtained by eliminating* b *in* Σ*: the clause* $a \vee d$ *of* Σ *which does not contain* b *is kept, and the resolvent* $a \vee c$ *of* $a \vee b$ *and* $\neg b \vee c$ *on* b *is conjoined to it.*

Observe that, by construction, the set of variables of $\exists \mathit{Var}(\Sigma).\Sigma$ is empty, so that the formula $\exists \mathit{Var}(\Sigma).\Sigma$ either is inconsistent or is valid, and this can be tested in linear time from $\exists \mathit{Var}(\Sigma).\Sigma$. Since $\exists \mathit{Var}(\Sigma).\Sigma$ is valid precisely when Σ is consistent, eliminating in a CNF formula Σ every variable occurring in it is a way to decide its satisfiability: it is the Davis-Putnam's algorithm for SAT [43] (we will return to it in Section 6).

Let us now recall the two (equivalent) forms under which the concept of definability in (classical) propositional logic can be encountered:

Definition 1 (Implicit Definability). *Let* $\Sigma \in \mathcal{L}_{\mathcal{P}}$, X *a finite subset of* $\mathcal{P}$, *and* $\mathrm{y} \in \mathcal{P}$. *The formula* Σ implicitly defines *the variable* y *in terms of* X *if and only if for every canonical term* γ_{X} *over* X, *we have* $\gamma_{\mathrm{X}} \wedge \Sigma \models \mathrm{y}$ *or* $\gamma_{\mathrm{X}} \wedge \Sigma \models \neg\mathrm{y}$.

Definition 2 (Explicit Definability). *Let* $\Sigma \in \mathcal{L}_{\mathcal{P}}$, X *a finite subset of* $\mathcal{P}$, *and* $\mathrm{y} \in \mathcal{P}$. *The formula* Σ explicitly defines *the variable* y *in terms of* X *if and only if there exists a formula* $\Phi_{\mathrm{X}} \in \mathcal{L}_{\mathcal{X}}$ *such that* $\Sigma \models (\Phi_{\mathrm{X}} \leftrightarrow \mathrm{y})$. *In such a case,* Φ_{X} *is called a* definition (or gate) of y on X in Σ, y *is the* output variable *of the gate, and* X *are its* input variables.

Let us illustrate the two notions of implicit definability and explicit definability using a simple example.

Example 4. *Let* Σ *be the* CNF *formula consisting of the following clauses:*

$\mathrm{a} \vee \mathrm{b}$,	$\neg\mathrm{a} \vee \neg\mathrm{b} \vee \mathrm{d}$,	$\mathrm{a} \vee \mathrm{e}$,
$\mathrm{a} \vee \mathrm{c} \vee \neg\mathrm{e}$,	$\neg\mathrm{a} \vee \neg\mathrm{c} \vee \mathrm{d}$,	$\mathrm{b} \vee \mathrm{c} \vee \mathrm{e}$,
$\mathrm{a} \vee \neg\mathrm{d}$,	$\neg\mathrm{a} \vee \neg\mathrm{b} \vee \mathrm{c} \vee \neg\mathrm{e}$,	$\neg\mathrm{b} \vee \neg\mathrm{c} \vee \mathrm{e}$.
$\mathrm{b} \vee \mathrm{c} \vee \neg\mathrm{d}$,	$\neg\mathrm{a} \vee \mathrm{b} \vee \neg\mathrm{c} \vee \neg\mathrm{e}$,	

Variables d *and* e *are implicitly defined in* Σ *in terms of* $\mathrm{X} = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$. *For instance, the canonical term* $\gamma_{\mathrm{X}} = \mathrm{a} \wedge \mathrm{b} \wedge \neg\mathrm{c}$ *is such that* $\gamma_{\mathrm{X}} \wedge \Sigma \models \mathrm{d} \wedge \neg\mathrm{e}$. *On the other hand,* $\gamma'_{\mathrm{X}} = \neg\mathrm{a} \wedge \neg\mathrm{b} \wedge \neg\mathrm{c}$ *is such that* $\gamma'_{\mathrm{X}} \wedge \Sigma$ *is inconsistent. Variables* d *and* e *are also explicitly defined in* Σ *in terms of* $\mathrm{X} = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ *since* Σ *implies*

$$\mathrm{d} \leftrightarrow [\mathrm{a} \wedge (\mathrm{b} \vee \mathrm{c})] \text{ and } \mathrm{e} \leftrightarrow [\neg\mathrm{a} \vee (\mathrm{b} \leftrightarrow \mathrm{c})].$$

What happens in this example is not fortuitous due to the following theorem:

Theorem 1 ([44]). *Let* $\Sigma \in \mathcal{L}_{\mathcal{P}}$, X *a finite subset of* $\mathcal{P}$, *and* $\mathrm{y} \in \mathcal{P}$. *The formula* Σ *implicitly defines the variable* y *in terms of* X *if and only if* Σ *explicitly defines* y *in terms of* X.

Since implicit definability and explicit definability coincide, one can simply say that y is defined in terms of X in Σ. More generally, we state that a subset Y of variables from $\mathcal{P}$ is defined in terms of X in Σ when every variable $y \in Y$ is defined in terms of X in Σ.

An interesting consequence of Theorem 1 is that it is not mandatory to point out a gate Φ_X of y on X in order to prove that such a gate exists. Indeed, it is enough to show that Σ implicitly defines y in terms of X to do the job, and this problem is "only" coNP-complete [33]. To prove it, we can take advantage of the following result (Padoa's theorem), restricted to propositional logic and recalled in [33]; this theorem gives an entailment-based characterization of (implicit) definability:

Theorem 2 ([45]). *For any $\Sigma \in \mathcal{L}_\mathcal{P}$ and X a finite subset of $\mathcal{P}$, let Σ'_X be the formula obtained by substituting in Σ every occurrence of a propositional variable z from $Var(\Sigma) \setminus X$ by a new propositional variable z'. Let $y \in \mathcal{P}$. If $y \notin X$, then Σ (implicitly) defines y in terms of X if and only if $\Sigma \wedge \Sigma'_X \wedge y \wedge \neg y'$ is inconsistent.*[2]

3. A New Preprocessing Technique for Model Counting

3.1. The B + E *Preprocessor*

Instead of detecting gates and replacing them in Σ in order to remove output variables, our preprocessing technique consists in detecting output variables, then in forgetting them in Σ. Thus, the first objective is to find a definability bipartition $\langle I, O \rangle$ of Σ. The notion of definability bipartition is given by:

Definition 3 (Definability Bipartition). *Let $\Sigma \in \mathcal{L}_\mathcal{P}$. A* definability bipartition *of Σ is a pair $\langle I, O \rangle$ such that $I \cup O = Var(\Sigma)$, $I \cap O = \emptyset$, and Σ defines every variable $o \in O$ in terms of I.*

The most interesting bipartitions are those for which I is "minimal" to some extent. Two notions of minimality can be considered here, since the minimality of I can be evaluated either at the set of variables itself, or as the cardinality of this set. Thus, a definability bipartition $\langle I, O \rangle$ of a formula Σ will be said to be

- a *subset-minimal* bipartition of Σ if and only if every pair $\langle I', O' \rangle$ such that $I' \cup O' = Var(\Sigma)$, $I' \cap O' = \emptyset$, and $I' \subset I$ is not a definability bipartition of Σ,

[2] Obviously enough, in the remaining case when $y \in X$, Σ defines y in terms of X.

- a *smallest* bipartition of Σ if and only if every pair $\langle I', O' \rangle$ such that $I' \cup O' = Var(\Sigma)$, $I' \cap O' = \emptyset$, and $\#(I') < \#(I)$ is not a definability bipartition of Σ.

Accordingly, $\langle I, O \rangle$ is a subset-minimal definability bipartition of Σ if and only if I is a minimal defining family (or base) for O with respect to Σ [33].

Clearly enough, every smallest bipartition of Σ is a subset-minimal bipartition of Σ, but not vice-versa. Furthermore, every formula Σ has a definability bipartition since $\langle I = Var(\Sigma), O = \emptyset \rangle$ is a bipartition of Σ. This bipartition is a smallest bipartition of Σ (hence a subset-minimal one) when every variable $y \in Var(\Sigma)$ is undefinable in Σ, which means that for every $X \subseteq \mathcal{P}$, Σ defines y in terms of X if and only if $y \in X$ [33]. Since $Var(\Sigma)$ is a finite set, this shows that every formula Σ has a smallest (hence a subset-minimal) bipartition.

Then, in a second step, the objective is to forget variables from O in Σ so as to simplify Σ. This leads to the two-step preprocessing algorithm B + E (*B(ipartition), then E(liminate)*) given at Algorithm 1, and reported here for the sake of completeness. The soundness of B + E is based on the following result:

Algorithm 1: B + E

input : a CNF formula Σ
output: a CNF formula Φ such that $\|\Phi\| = \|\Sigma\|$

```
1 O←B(Σ);
2 Φ←E(O, Σ);
3 return Φ
```

Proposition 1. *Let* $\Sigma \in \mathcal{L}_{\mathcal{P}}$. *Let* $\langle I, O \rangle$ *be a definability bipartition of* Σ. *Let* $E \subseteq O$. *Then* $\|\Sigma\| = \|\exists E.\Sigma\|$.

Proof: Let $E = \{y_1, \ldots, y_m\}$ be a subset of O. Since every y_i ($i \in 1, \ldots, m$) is definable in terms of I in Σ, there exists a formula $\Phi_I^{y_i}$ over I such that

$$\Sigma \models (y_i \leftrightarrow \Phi_I^{y_i})$$

(i.e., a gate $\Phi_I^{y_i}$ of y_i on I in Σ exists).

Let $\Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i \in 1,\ldots,m}$ be the formula obtained by replacing in Σ every occurrence of y_i by $\Phi_I^{y_i}$. Let γ_I be a canonical term over I. If $\gamma_I \wedge \Sigma$ is consistent, then there exists a unique model ω_{γ_I} of Σ over $Var(\Sigma)$ that is a model of γ_I. Accordingly, every model of Σ is fully characterized by its restriction over I, so that $\|\Sigma\|$ is equal to the number of canonical terms γ_I over I such that $\gamma_I \wedge \Sigma$ is consistent.

Now, by construction, we have

$$\Sigma \equiv \bigwedge_{i=1}^{m}(y_i \leftrightarrow \Phi_I^{y_i}) \wedge \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}.$$

Hence $\gamma_I \wedge \Sigma$ is equivalent to

$$\gamma_I \wedge \bigwedge_{i=1}^{m}(y_i \leftrightarrow \Phi_I^{y_i}) \wedge \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}.$$

Since γ_I is a canonical term over I and $\mathit{Var}(\Phi_I^{y_i}) \subseteq I$ for every $i \in 1,\ldots,m$, we have that $\gamma_I \wedge \Phi_I^{y_i}$ is consistent if and only if $\gamma_I \models \Phi_I^{y_i}$, so that $\gamma_I \wedge \bigwedge_{i=1}^{m}(y_i \leftrightarrow \Phi_I^{y_i})$ is equivalent to $\gamma_I \wedge \bigwedge_{i=1}^{m} y_i^*$ where y_i^* ($i \in 1,\ldots,m$) is y_i when $\gamma_I \models \Phi_I^{y_i}$ and is $\neg y_i$ otherwise. Therefore,

$$\gamma_I \wedge \bigwedge_{i=1}^{m}(y_i \leftrightarrow \Phi_I^{y_i}) \wedge \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}$$

is equivalent to

$$\gamma_I \wedge (\bigwedge_{i=1}^{m} y_i^*) \wedge \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}.$$

In addition, since $\mathit{Var}(\gamma_I \wedge \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}) \subseteq I$, $\mathit{Var}(\bigwedge_{i=1}^{m} y_i^*) \subseteq O$ and $I \cap O = \emptyset$, $\gamma_I \wedge \Sigma$ is consistent if and only if $\gamma_I \wedge \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}$ is consistent. But since γ_I is a canonical term over I and $\mathit{Var}(\Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}) \subseteq I$, this is precisely the case when $\gamma_I \models \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}$. Thus the number of canonical terms γ_I over I such that $\gamma_I \wedge \Sigma$ is consistent is equal to the number of models of $\Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}$ over I. Stated otherwise, we have $\|\Sigma\| = \|\Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}\|$.

Since $\mathit{Var}(\Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}) \cap E = \emptyset$, we have that $\exists E.\Sigma$ which is equivalent to

$$\exists E.((\bigwedge_{i=1}^{m}(y_i \leftrightarrow \Phi_I^{y_i}) \wedge \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}),$$

is also equivalent to

$$(\exists E.(\bigwedge_{i=1}^{m}(y_i \leftrightarrow \Phi_I^{y_i}))) \wedge \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i\in 1,\ldots,m}.$$

Finally, since $Var(\Phi_I^{y_i}) \cap E = \emptyset$, we also have that $\exists E.(\bigwedge_{i=1}^{m}(y \leftrightarrow \Phi_I^{y_i}))$ is equivalent to $\bigwedge_{i=1}^{m}(\exists y_i.(y_i \leftrightarrow \Phi_I^{y_i}))$. But each $\exists y_i.(y_i \leftrightarrow \Phi_I^{y_i})$ $(i \in 1, \ldots, m)$ is a valid formula. Hence we have

$$\exists E.\Sigma \equiv \Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i \in 1,\ldots,m},$$

which implies that

$$\|\exists E.\Sigma\| = \|\Sigma[y_i \leftarrow \Phi_I^{y_i}]_{i \in 1,\ldots,m}\|,$$

and thus that $\|\Sigma\| = \|\exists E.\Sigma\|$. □

Example 5 (Example 4 cont'ed). *No literal equivalences, AND/OR gates or XOR gates are logical consequences of* Σ*. Nevertheless, since* Σ *implies*

$$d \leftrightarrow [a \wedge (b \vee c)] \textit{ and } e \leftrightarrow [\neg a \vee (b \leftrightarrow c)]$$

a definability bipartition of Σ *is* $\langle \{a, b, c\}, \{d, e\} \rangle$*. Now, eliminating* d *and* e *in* Σ *using the resolution principle leads to the generation of two clauses* $a \vee c$ *and* $a \vee b \vee c$ *(the other resolvents that are produced are valid clauses, hence they can be omitted). Therefore, a* CNF *formula equivalent to* $\exists\{d, e\}.\Sigma$ *can be computed as the conjunction of:*

$a \vee b$, $a \vee c$, $a \vee b \vee c$,

which can be simplified further into $(a \vee b) \wedge (a \vee c)$*. This* CNF *formula has only* 5 *models over* $\{a, b, c\}$*. From Proposition 3, this is also the case of* Σ *over* $\{a, b, c, d, e\}$.

The ability to identify any subset O of the full set of output variables in the bipartition generation phase, and to consider only a subset E of O in the elimination phase are two important features for the efficiency purpose.

On the one hand, computing a subset-minimal definability bipartition of Σ turns out to be computationally easier than computing a smallest definability bipartition of Σ. Indeed, given a definability bipartition $\langle I, O \rangle$ of Σ, determining whether it is a subset-minimal one does not require to check that each of the (exponentially many) subsets of I does not define in Σ every variable of $Var(\Sigma)$. To be more precise, it is enough to consider only the (linearly many) subsets $I \setminus \{x\}$ with $x \in I$ (see the proof of Proposition 2). Thus, only one source of complexity (coming from the definability test) must be dealt with when one looks for a subset-minimal definability bipartition. Formally:

Definition 4 (SUBSET-MINIMAL BIPARTITION).
SUBSET-MINIMAL BIPARTITION *is the following decision problem:*

- *Input: a* CNF *formula* Σ*, and a definability bipartition* $\langle \mathrm{I}, \mathrm{O} \rangle$ *of* Σ*.*
- *Question: is* $\langle \mathrm{I}, \mathrm{O} \rangle$ *a subset-minimal bipartition of* Σ*?*

Proposition 2. SUBSET-MINIMAL BIPARTITION *is* NP*-complete.*

Proof:

- Membership in NP. We first show that a given definability bipartition $\langle \mathrm{I}, \mathrm{O} \rangle$ of Σ is not a subset-minimal one if and only if there exists $\mathrm{x} \in \mathrm{I}$ such that $\langle \mathrm{I} \setminus \{\mathrm{x}\}, \mathrm{O} \cup \{\mathrm{x}\} \rangle$ is a definability bipartition of Σ, which is precisely the same as stating that there exists $\mathrm{x} \in \mathrm{I}$ such that $\mathrm{I} \setminus \{\mathrm{x}\}$ defines in Σ every variable of $Var(\Sigma)$ (1). Clearly, by definition, a definability bipartition $\langle \mathrm{I}, \mathrm{O} \rangle$ of Σ is not a subset-minimal one if and only if there exists $\mathrm{J} \subset \mathrm{I}$ such that J defines in Σ every variable of $Var(\Sigma)$ (2). Obviously, (1) implies (2) (just take $\mathrm{J} = \mathrm{I} \setminus \{\mathrm{x}\}$). Conversely, we also have that (2) implies (1), due to the monotonicity property offered by definability [33]. This property states that if J defines in Σ every variable of $Var(\Sigma)$, then every superset of J also defines in Σ every variable of $Var(\Sigma)$. Indeed, if (2) holds, then for any $\mathrm{J} \subset \mathrm{I}$ such that J defines in Σ every variable of $Var(\Sigma)$, there exists $\mathrm{x} \in \mathrm{I}$ such that $\mathrm{J} \subseteq \mathrm{I} \cup \{\mathrm{x}\}$; thanks to the monotonicity property, we get that for any such J, the superset $\mathrm{I} \cup \{\mathrm{x}\}$ of J also defines in Σ every variable of $Var(\Sigma)$, and (1) is satisfied.

  As a consequence, a given definability bipartition $\langle \mathrm{I}, \mathrm{O} \rangle$ of Σ is a subset-minimal bipartition of Σ if and only if $\mathrm{O} \cup \{\mathrm{x}\}$ is not defined in terms of $\mathrm{I} \setminus \{\mathrm{x}\}$ in Σ whatever $\mathrm{x} \in \mathrm{I}$. Thus, deciding whether a given definability bipartition $\langle \mathrm{I}, \mathrm{O} \rangle$ of Σ is a subset-minimal bipartition of Σ amounts to solving #(I) independent instances (one for each possible $\mathrm{x} \in \mathrm{I}$) of the NON-DEFINABILITY problem given by

  - Input: a formula Σ, and two sets of variables $\mathrm{I} \setminus \{\mathrm{x}\}$ and $\mathrm{O} \cup \{\mathrm{x}\}$.
  - Question: is $\mathrm{O} \cup \{\mathrm{x}\}$ not defined in terms of $\mathrm{I} \setminus \{\mathrm{x}\}$ in Σ?

  The complementary problem to NON-DEFINABILITY, namely DEFINABILITY, has been shown coNP-complete [33], hence NON-DEFINABILITY $\in$

NP. Consequently, the problem consisting of deciding given $\langle \Sigma, I, O \rangle$ whether $\langle \Sigma, I \setminus \{x\}, O \cup \{x\} \rangle$ (where x belongs to I) belongs to NON-DEFINABILITY is in NP as well.

Let REPEATED-NON-DEFINABILITY be the language defined as the union for all integers $k > 0$ up to $\#(\mathcal{P})$ (the number of variables in the language $\mathcal{L}_{\mathcal{P}}$) of the languages k-REPEATED-NON-DEFINABILITY taking as inputs k-tuples (where k is fixed) gathering instances of the NON-DEFINABILITY problem.

Consider now the following mapping: with any instance of SUBSET-MINIMAL BIPARTITION given by Σ, $I = \{x_1, \ldots, x_k\}$, and O, we associate the k-tuple $\langle \langle \Sigma, I \setminus \{x_1\}, O \cup \{x_1\} \rangle, \ldots, \langle \Sigma, I \setminus \{x_k\}, O \cup \{x_k\} \rangle \rangle$ where each $\langle \Sigma, I \setminus \{x_i\}, O \cup \{x_i\} \rangle$ ($i \in \{1, \ldots, k\}$) is an instance of the NON-DEFINABILITY problem. This mapping is a polynomial-time many-one reduction from SUBSET-MINIMAL BIPARTITION to REPEATED-NON-DEFINABILITY.

Since the Cartesian product of languages from NP is in NP, for any finite k, k-REPEATED-NON-DEFINABILITY belongs to NP. Hence, the finite union REPEATED-NON-DEFINABILITY of such languages belongs to NP as well. Finally, since NP is closed under polynomial-time many-one reductions, SUBSET-MINIMAL BIPARTITION belongs to NP, and this concludes the proof (membership part).

- NP-hardness. One reduces SAT, the satisfiability problem for CNF formulae, which is the canonical NP-complete problem [46], to SUBSET-MINIMAL BIPARTITION. Let α be a CNF formula. We associate with it in polynomial time the instance given by $\Sigma = \alpha \vee \mathit{new}$ where *new* is a fresh variable (not occurring in $\mathit{Var}(\alpha)$), $I = \mathit{Var}(\Sigma)$ and $O = \emptyset$. Clearly, $\langle I, O \rangle$ is a definability bipartition of Σ. We now show that it is a subset-minimal bipartition of Σ precisely when α is consistent. Consider first any variable $x \in \mathit{Var}(\alpha)$. $(\Sigma \mid x) \wedge (\Sigma \mid \neg x)$ is consistent since every interpretation satisfying *new* is a model of it. Thus x is undefinable in Σ. Hence, in any definability bipartition of Σ, $\mathit{Var}(\alpha)$ must be a subset of the set of input variables. Furthermore, $(\Sigma \mid \mathit{new}) \wedge (\Sigma \mid \neg \mathit{new}) \equiv \alpha$. Hence, *new* is undefinable in Σ precisely when α is consistent. Thus, if α is consistent, then $\langle \mathit{Var}(\Sigma), \emptyset \rangle$ is the unique definability bipartition of Σ; therefore, it is a subset-minimal bipartition of Σ. In the remaining case when α is inconsistent, we have $\Sigma \equiv \mathit{new}$, thus $\langle \mathit{Var}(\alpha), \{\mathit{new}\} \rangle$ is a definability bipartition of Σ, showing

that $\langle Var(\Sigma), \emptyset \rangle$ is not a subset-minimal bipartition of Σ.

□

Clearly enough, the complexity of SUBSET-MINIMAL BIPARTITION identified in Proposition 2 coheres with Theorem 24 from [33], showing that checking whether a given set X of variables is a minimal defining family for a variable y with respect to Σ is NP-complete.

On the other hand, while forgetting variables in Σ obviously leads to reducing the number of variables occurring in it, it may also lead to an exponential increase of its size. This is why one refrains from eliminating in the `CNF` formula Σ every variable of O but focuses instead on a subset E of O, containing those variables for which the elimination step will not increase the size of Σ (similar to what is done with the NiVER approach [20]), or only by a negligible factor. More generally, the elimination of an output variable from O is committed (i.e., this variable is considered to belong to E) only if the size of Σ after the elimination of this variable in Σ remains small enough, once some additional preprocessing techniques have been applied. Among the equivalence-preserving preprocessing techniques of interest are occurrence simplification [21] and vivification [23] (already considered in [32]), which aim to shorten some clauses (for occurrence simplification and vivification), and to remove some clauses (for vivification). The removal of subsumed resolvents can also be achieved at each step.

Let us now detail successively the two steps of `B` + `E`, namely `B` (computing a bipartition $\langle I, O \rangle$ of Σ), and `E` (eliminating in Σ some variables from O).

3.2. `B`*(ipartition)*

Algorithm 2 shows how a bipartition $\langle I, O \rangle$ of $Var(\Sigma)$ is computed by `B` in a greedy fashion. At line 1, `backbone`(Σ) computes the backbone of Σ (i.e., the set of all literals implied by Σ). This backbone is computed using the algorithm `backboneSimpl` reported in [32]. `backbone`(Σ) also initializes O with the variables of the backbone of Σ (indeed, a literal ℓ belongs to the backbone of Σ precisely when $var(\ell)$ is defined in Σ in terms of $\emptyset$). Boolean constraint propagation is also done on Σ completed by its backbone (this typically leads to simplifying Σ). While the variables of the backbone can be simplified away in Σ by fixing their values, they are nevertheless kept in O in order to ensure that the set O of variables returned by `B` is such that $\langle I, O \rangle$ is a bipartition of $Var(\Sigma)$. At line 2, the set V of remaining variables occurring in Σ (after simplification) is sorted by considering their number of occurrences from less to more frequent. The rationale

Algorithm 2: B

input : a CNF formula Σ

output: a set O of output variables, i.e., variables defined in Σ in terms of $I = Var(\Sigma) \setminus O$

```
1 ⟨Σ, O⟩ ← backbone(Σ);
2 V ← sort(Var(Σ));
3 I ← ∅;
4 foreach x ∈ V do
5     if defined?(x, Σ, I ∪ succ(x, V), max#C) then
6         O ← O ∪ {x};
7     else
8         I ← I ∪ {x};
9 return O
```

for this ordering heuristics lies in the fact that if x is not very frequent in Σ, then it is not linked to many other variables, so that the likelihood of Σ to define x in terms of a small number of variables is supposed to be low. Accordingly, when x is a least frequent variable in Σ, one tests first whether Σ defines or not x in terms of all the other variables, which is the most favorable case for classifying x as an output variable (this is due to the monotonicity of the definability relation with respect to the set of input variables, i.e., the fact that if Σ defines x in terms of a set X of variables, then Σ also defines x in terms of any superset of X [33]). At line 4, `defined?` takes advantage of Padoa's method (Theorem 2) for determining whether x is defined in Σ in terms of $I \cup \texttt{succ}(x, V)$, where $\texttt{succ}(x, V)$ is the set of all variables of V that appear after x in V. `defined?` takes advantage of a SAT solver `solve` based on a conflict-driven clause learning (CDCL) architecture (see [47, 48, 49]) for achieving the (un)satisfiability test required by Padoa's method. In our implementation, the input of `solve` is the CNF formula $\Sigma \wedge \Sigma'_{\emptyset} \wedge \bigwedge_{z \in Var(\Sigma)}((\neg s_z \vee \neg z \vee z') \wedge (\neg s_z \vee z \vee \neg z'))$, completed by *assumptions*: for every z belonging to $I \cup \texttt{succ}(x, V)$, the unit clause s_z associated with z is added as an assumption to the CNF formula (its effect is to make z equivalent to its copy z'); then, x and $\neg x'$ are also added as assumptions. Interestingly, clauses that are learnt at each call to `solve` are kept for the subsequent calls. `defined?` is parameterized by `max#C` which bounds the number of clauses that can be learnt. When no contradiction has been found before `max#C` is reached, `defined?` returns false

(i.e., x is considered as not defined in Σ in terms of $I \cup \texttt{succ}(x, V)$, while this could be questioned had a larger bound be considered).

Clearly enough, the number of output variables found by B cannot be guaranteed to be as large as possible (or equivalently, the number of input variables found by B is not guaranteed to be as small as possible). This is due to the fact that the insertion of a variable in I at line 8 of B cannot be questioned by the execution of a subsequent instruction of this algorithm. But, as already explained, this lack of optimality is on purpose, for the sake of efficiency. Indeed, in B, the number of calls to solve does not exceed the number of variables occurring in Σ. Nevertheless, one can ensure that :

Proposition 3. *Let Σ be a* CNF *formula. The set* O *computed by* B *is such that $\langle I, O\rangle$ where $I = Var(\Sigma) \setminus O$ is a subset-minimal definability bipartition of Σ.*

Proof: First of all, every variable x of $Var(\Sigma)$ such that x or $\neg x$ belongs to the backbone of Σ is such that x is defined in Σ in terms of $\emptyset$. Thus, any subset-minimal definability bipartition $\langle I, O\rangle$ of Σ must guarantee that those variables x belong to O. This is ensured in B by the instruction at line 1.

Let us now focus on the remaining set V of variables. In B, the variables $x \in V$ are considered from the ones having the smallest number of occurrences in Σ (after simplification) to those having the greatest number of occurrences in Σ (after simplification). However, the result stated in Proposition 3 actually holds whatever the ordering $<$ with respect to which the variables of V are visited. Thus, we prove by induction on $k = \#(\texttt{succ}(x, V))$ that every variable x put into O at line 6 of B is defined in Σ in terms of I. The base case is when $k = 0$ so that x is the last variable of V with respect to $<$. In that case, $\texttt{succ}(x, V) = \emptyset$ so that x is put into O when the test at line 5 is evaluated to true, i.e., when x is defined in Σ in terms of I, and we are done. Suppose now that the property holds for $k = p \geq 1$ and let us show that it still holds for $k = p + 1$. So, let x be the variable of V such that $\#(\texttt{succ}(x, V)) = p + 1$. Line 5 of B ensures that x is put into O when the corresponding test is evaluated to true. At the step when x is processed, the current value of I is the set I_x which contains the variables that will be in I at the end of the execution of B and are before x with respect to $<$. Thus, assuming that $\texttt{prec}(x, V)$ denotes the set of variables of V that are considered before x with respect to $<$, we have $I_x = I \cap \texttt{prec}(x, V)$ and x is put into O when $\texttt{defined?}(x, \Sigma, I_x \cup \texttt{succ}(x, V), \texttt{max\#C})$ is evaluated to true, i.e., when x is defined in Σ in terms of $I_x \cup \texttt{succ}(x, V)$. But $I_x \cup \texttt{succ}(x, V) = (I \cap \texttt{prec}(x, V)) \cup \texttt{succ}(x, V) = (I \cap \texttt{prec}(x, V)) \cup (I \cap \texttt{succ}(x, V)) \cup (O \cap$

$\texttt{succ}(x, V)) = (I \cap (V \setminus \{x\})) \cup (O \cap \texttt{succ}(x, V))$. Furthermore, if x is put in O, then x does not belong to I. Therefore, in this case, we have that x is defined in Σ in terms of $I_x \cup \texttt{succ}(x, V) = I \cup (O \cap \texttt{succ}(x, V))$. Now, by induction hypothesis, every variable $y \in \texttt{succ}(x, V)$ belongs to O if and only if y is defined in Σ in terms of I. Thus, all the variables y from $O \cap \texttt{succ}(x, V)$ can be removed from $I \cup (O \cap \texttt{succ}(x, V))$ while preserving the fact that x is defined in Σ. We get that if x is put into O at line 6 of B, then x is defined in Σ in terms of I. This implies that every variable in O is defined in Σ in terms of I, hence the pair $\langle I, O \rangle$ returned by B is a definability bipartition of Σ.

Finally, it remains to prove that this definability bipartition $\langle I, O \rangle$ is subset-minimal. Towards a contradiction, suppose that there exists $x \in I$ such that $\langle I \setminus \{x\}, O \cup \{x\} \rangle$ is a definability bipartition of Σ. Since x has been put into I by B, it must be the case that $\texttt{defined?}(x, \Sigma, I_x \cup \texttt{succ}(x, V), \texttt{max\#C})$ has been evaluated to false, i.e., x is not defined in Σ from $(I \cap \texttt{prec}(x, V)) \cup \texttt{succ}(x, V)$. Furthermore, we have that $I \setminus \{x\} = I \cap (\texttt{prec}(x, V) \cup \texttt{succ}(x, V)) = (I \cap \texttt{prec}(x, V)) \cup (I \cap \texttt{succ}(x, V)) \subseteq (I \cap \texttt{prec}(x, V)) \cup \texttt{succ}(x, V)$. However, if x is not defined in Σ from $(I \cap \texttt{prec}(x, V)) \cup \texttt{succ}(x, V)$, then x cannot be defined in Σ from a subset $I \setminus \{x\}$ of it. This contradicts the fact that $\langle I \setminus \{x\}, O \cup \{x\} \rangle$ is a definability bipartition of Σ. □

Note that it would be easy to derive an "anytime" version of B without questioning the fact that its output O is such that $\langle I = \mathit{Var}(\Sigma) \setminus O, O \rangle$ is a definability bipartition of Σ. Indeed, when the time limit under consideration would be reached, it would be enough to put into I every variable that has not been put into O at this point.

Interestingly, the correctness of B (as given by Proposition 3) does not require any assumption on the representation of Σ (i.e., Σ is not necessarily a CNF formula from $\mathcal{L}_{\mathcal{P}}$). Thus, the approach followed (i.e., find a definability bipartition of Σ, then eliminate some output variables) is sound when the input is not a CNF formula, and in such a case, it can be less demanding from a computational standpoint than the case when CNF inputs are considered. Consider for instance the case when Σ is a DNF formula. While computing $\|\Sigma\|$ is still #P-complete when Σ is a DNF formula, the definability problem (determine whether Σ defines a given x in terms of a given X) and the variable elimination problem can be solved in (deterministic) polynomial time (see Lemma 27 in [33] and Propositions 17 and 18 in [42]). Thus, when Σ is a DNF formula, it makes sense to consider a more aggressive strategy for computing a definability bipartition (since each definability test is not very time consuming, one can try to minimize the cardinality of I), and to eliminate every variable from O in the variable elimination step (this can be done

in linear time in the size of Σ and can reduce significantly the size of Σ). However, this does not imply that exploiting duality and computing $\|\Sigma\|$ for a CNF formula Σ as $2^{\#Var(\Sigma)} - \|\neg\Sigma\|$ is always a good idea from a computational point of view. Indeed, even if a DNF formula equivalent to $\neg\Sigma$ can be computed in time linear in the size of Σ, it is in general not the case that Σ and $\neg\Sigma$ define the same variables. Thus, it can easily be the case that a definability bipartition of Σ with a "small" I exists, while every variable of $Var(\Sigma)$ is undefinable in $\neg\Sigma$. For instance, consider $\Sigma = \bigwedge_{i=1}^{n} x_i$. Every x_i ($i \in \{1, \ldots, n\}$) belongs to the backbone of Σ so that $\langle \emptyset, Var(\Sigma) \rangle$ is a definability bipartition of Σ such that $I = \emptyset$ is of smallest cardinality; conversely, every x_i ($i \in \{1, \ldots, n\}$) is undefinable in $\neg\Sigma$ which is equivalent to $\bigvee_{i=1}^{n} \neg x_i$.

3.3. E*(liminate)*

Algorithm 3 shows how variables from O are eliminated in Σ by E. P contains variables x from O that are candidates for elimination, so that the elimination of x is possibly postponed in the process. P is initialized with the full set O (line 2). The main loop at line 3 is repeated by considering the variables from P as candidates for elimination, while the elimination of at least one variable is effective (line 16). At line 4, the set E of variables that will be tentatively eliminated during the iteration is initialized with P, and P is reset to $\emptyset$. At line 5, the clauses of Φ are successively vivified (i.e., one tries to remove them and/or to shorten them) using a slight variant of the vivification algorithm `vivificationSimpl` reported in [32]. Vivification [23] is a preprocessing technique which aims to reduce the input CNF formula, i.e., to remove some clauses in it and some literals in the other clauses while preserving equivalence, using Boolean constraint propagation. To be more precise, given a clause $\delta = \ell_1 \vee \ldots \vee \ell_k$ of Σ, two rules are used in order to determine whether δ can be removed from Σ or simply shortened. Thus, for each clause δ of Σ the literals ℓ_{j+1} of δ are successively considered and the question is to determine whether they should be added or not to the current subclause $\delta' = \ell_1 \vee \ldots \vee \ell_j$ of δ (where δ' is initialized as the empty clause). On the one hand, if for any $j \in \{0, \ldots, k-1\}$, one can prove using Boolean constraint propagation that $\Sigma \setminus \{\delta\} \models \delta'$, then for sure δ is entailed by $\Sigma \setminus \{\delta\}$ so that δ can be removed from Σ. On the other hand, if one can prove using Boolean constraint propagation that $\Sigma \setminus \{\delta\} \models \delta' \vee \sim\ell_{j+1}$, then ℓ_{j+1} can be removed from δ without questioning equivalence. The additional parameter E is used to sort the literals within the clauses of Σ so that the literals over E are put first (i.e., one tries to eliminate occurrences of literals over E in priority). At line 6, one enters into the inner loop that operates while there are remaining variables in E. At line 7,

Algorithm 3: `E`

input : a `CNF` formula Σ and a set of output variables $O \subseteq \mathit{Var}(\Sigma)$
output: a `CNF` formula Φ such that $\Phi \equiv \exists E.\Sigma$ for some $E \subseteq O$

```
1  Φ←Σ;
2  iterate←true; P←O;
3  while iterate do
4      E←P; P←∅; iterate←false;
5      Φ←vivificationSimpl(Φ, E);
6      while E ≠ ∅ do
7          x←select(E, Φ);
8          E←E \ {x};
9          Φ←occurrenceSimpl(Φ, x);
10         if #(Φ_x) × #(Φ_¬x) > max#Res then
11             P←P ∪ {x}
12         else
13             R←removeSub(Res(x, Φ), Φ);
14             if #((Φ \ Φ_{x,¬x}) ∪ R) ≤ #(Φ) then
15                 Φ←(Φ \ Φ_{x,¬x}) ∪ R;
16                 iterate←true;
17             else
18                 P←P ∪ {x}
19 return Φ
```

a variable x is selected in E for being possibly eliminated by counting the number $\#(\Phi_x)$ of clauses of Φ where x appears as a positive literal, and the number $\#(\Phi_{\neg x})$ of clauses of Φ where $\neg x$ appears as a negative literal; x is retained if it minimizes $\#(\Phi_x)\times\#(\Phi_{\neg x})$, which is an upper bound of the number of resolvents that the elimination of x in Φ may generate. At line 8, x is removed from E. Then, at line 9, one tries first to eliminate in Φ some occurrences of variable x using `occurrenceSimpl`. `occurrenceSimpl` is a restriction of the algorithm for occurrence simplification reported in [32], where instead of considering the whole set of literals occurring in Φ, we just focus on those in $\{x, \neg x\}$. At line 10, one recomputes $\#(\Phi_x)\times\#(\Phi_{\neg x})$ and checks whether it exceeds or not a preset bound `max#Res`. If this is the case, then we possibly postpone the elimination of x in Φ at the next iteration by adding it to P (line 11). Otherwise, we compute the set `Res`(x, Φ) of all non-valid resolvents of clauses from Φ on x and we remove from it using `removeSub` every clause that is properly subsumed by a clause of Φ or another clause from `Res`(x, Φ); the resulting set of clauses is R (line 13). At line 14, we test whether the elimination of x in Φ, obtained by removing from Φ its subset $\Phi_{x,\neg x}$ of the clauses into which variable x occurs (either as a positive literal or as a negative literal), and adding the resolvents from R, leads or not to increasing the number of clauses in Φ. If so, then we possibly postpone the elimination of x in Φ at the next iteration by adding it to P (line 18). If not, the elimination of x in Φ is committed (line 15).

The worst-case time complexity of Algorithm `E` is in $\mathcal{O}(|O|^2 \cdot |\Sigma|^3)$. Indeed, the inner loop is executed only if at least one variable has been eliminated at the previous step, hence every variable of O cannot be considered more than a quadratic number of times as a candidate for being eliminated. The worst-case time complexity of Boolean constraint propagation is linear in the input size [50], but quadratic when the set of clauses considered by `solve` is implemented using watched literals [49], which is the case in our implementation.[3] Under this assumption, the vivification algorithm used in the outer loop has a time complexity that is cubic in its input size, and the occurrence simplification algorithm used in the inner loop has a time complexity that is cubic in the size of its input. At each step, the size of the `CNF` formula under consideration can only increase by a constant term, which can be neglected.

It is easy to show that Algorithm `E` is correct:

[3] In practice, it typically runs in sublinear time in the input size.

Proposition 4. *Let* Σ *be a* CNF *formula and* O *be a set of variables of* Σ*. The* CNF *formula* Φ *computed by* E *is such that* $\Phi \equiv \exists \mathrm{E}.\Sigma$ *for some* $\mathrm{E} \subseteq \mathrm{O}$*.*

Proof: The result comes directly from three observations:
(1) the fact that `vivificationSimpl` is an equivalence-preserving preprocessing technique, (2) the variables x that are eliminated belong to O (see lines 2, 4 and 7 of `E`), and (3) $\exists \mathrm{x}.\Sigma$ is equivalent to the formula obtained by adding first to Σ all resolvents on x of the clauses of Σ, then removing in the resulting set all the clauses containing x as a variable. This is performed by the instructions at lines 13 and 15 of `E` (the additional suppression of valid or subsumed resolvents achieved by `removeSub` is harmless since it is equivalence-preserving). □

Again, it would be easy to derive an "anytime" version of `E` without questioning its correctness. Indeed, when the time limit under consideration would be reached, it would be enough to stop the elimination process at this point (i.e., not considering the variables of O that are in E or in P).

4. Empirical Results

Let us now present the experiments which have been done for evaluating `B` + `E` and comparing it with `pmc`, our previous preprocessor for model counting.

4.1. Empirical Setting

In our experiments, we have considered 703 `CNF` instances from the SATLIB[4] and other repositories (for instance, the benchmarks from the BN family (Bayesian networks) come from `http://reasoning.cs.ucla.edu/ace/`). All the benchmarks used in our experiments can be downloaded from `http://www.cril.univ-artois.fr/KC/bpe2.html`. They are gathered into 8 data sets, as follows: BN (192), BMC (Bounded Model Checking) (18), Circuit (41), Configuration (35), Handmade (58), Planning (248), Random (104), Qif (7) (Quantitative Information Flow analysis - security).

All the experiments have been conducted on a cluster of Intel Xeon E5-2643 (3.30 GHz) quad core processors with 32 GiB RAM. The kernel used was CentOS 7, Linux version 3.10.0-514.16.1.el7.x86_64. The compiler used was gcc version 5.3.1. Hyperthreading was disabled, and no cache share between cores was allowed. A time-out of 1h and a memory-out of 7.6 GiB has been considered for

[4] `www.cs.ubc.ca/~hoos/SATLIB/index-ubc.html`

each instance. We set max#Res to 500. We took advantage of the runsolver[5] tool [51] for controlling the resources used by the SAT solver solve used in our experiments, namely the incremental version of Glucose described in [52] (run with its default setting).

As a matter of comparison, we have considered the pmc preprocessor for model counting, described in [32] and available on www.cril.fr/KC/. To be more precise, we considered pmc equipped with the *#eq* combination of preprocessing techniques, which combines backbone simplification, occurrence elimination, vivification and gates detection and replacement. pmc equipped with *#eq* proved empirically as a very efficient preprocessor for model counting [32].

We evaluated the impact of B + E (for several values of max#C) by coupling it with exact model counters. We considered the search-based model counters Cachet[6] [53] and SharpSAT[7] [54], run with their default settings. Though compilation-based approaches do much more than model counting (since they compute equivalent, compiled representations of the input CNF formula Σ and not only the number of models of Σ), some of them appear as competitive for the model counting purpose. Thus, we also took advantage of such compilers, namely C2D[8] [55, 56] and d4[9] [37]. Those compilers generate a Decision-DNNF representation Σ^* of the input Σ. The size of Σ^* is exponential in the size of Σ in the worst case, but the number of models of Σ conditioned by any consistent term γ can be computed efficiently from Σ^* in every case. And when γ is $\top$, one gets the number of models of Σ. C2D has been invoked with the following options -count -in_memory -smooth_all, which are suited when C2D is used as a model counter. d4 has been invoked with its default options.

4.2. Results

Table 1 makes precise the number of instances (out of 703) solved within 1h by each of the model counters Cachet, SharpSAT, C2D and d4 (first column), when no preprocessing technique has been applied (second column), pmc (equipped with *#eq*) has been applied first (third column), and finally B + E(Σ) for several values of max#C has been applied first (the remaining columns). The

[5] www.cril.fr/~roussel/runsolver
[6] www.cs.rochester.edu/~kautz/Cachet/
[7] sites.google.com/site/marcthurley/sharpsat
[8] reasoning.cs.ucla.edu/c2d/
[9] www.cril.univ-artois.fr/KC/d4.html

preprocessing time is taken into account in the computations (it is part of the 1h CPU time allocated per instance).

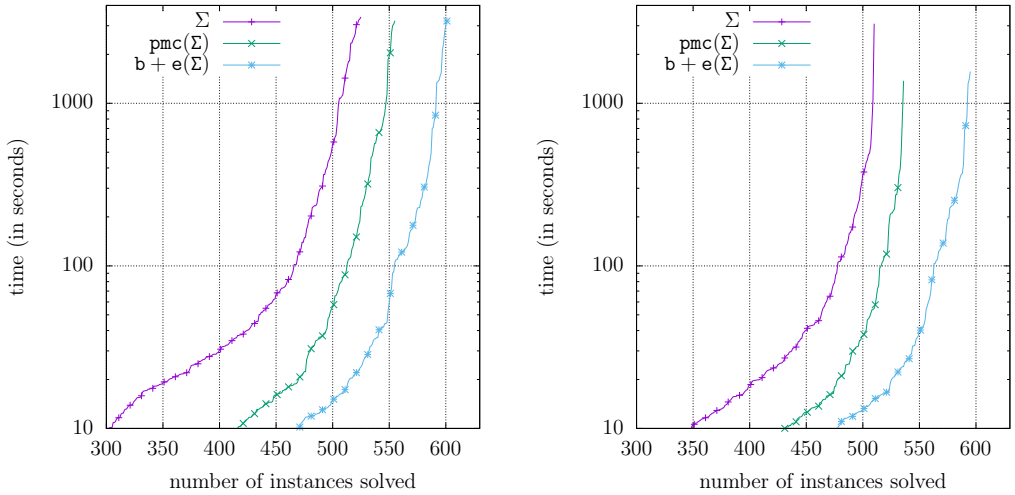
model counter	no preproc.	pmc	10	100	1000	∞
Cachet	525	558	586	588	594	602
SharpSAT	507	537	575	581	586	593
C2D	547	602	605	613	616	625
d4	583	607	614	615	617	622

Table 1: Number of instances solved within the time limit depending on the preprocessing technique used.

The results reported in Table 1 show the benefits that can be achieved by applying B + E before using a model counter. Globally speaking, B + E leads to better performance than pmc. Since the best performances of B + E are achieved for max#C = ∞, we focus on this parameter assignment in the rest of the paper.

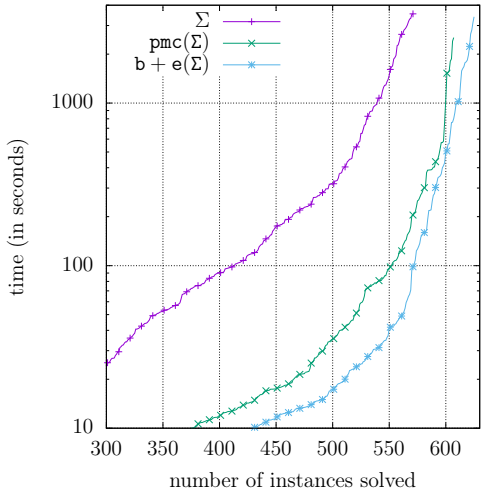
The cactus plots given in Figure 1 illustrate the performances of Cachet (Fig. 1a), SharpSAT (Fig. 1b), C2D (Fig. 1c) and d4 (Fig. 1d), possibly empowered by pmc or by B + E. For each value t on the y-axis (a model counting time, in seconds) and each dot of a curve for which this value is reached on the y-axis, the corresponding value on the x-axis makes precise how many instances have been solved by the approach associated with the curve within a time limit of t (which includes the preprocessing time, when a preprocessing technique has been used). For the sake of readability, only 10% of the dots have been printed. Again, the plot clearly shows that preprocessing the instances is computationally valuable whatever the downstream model counter, and that B + E is typically a better preprocessor than pmc.

Table 2 aims to give a more precise view of the importance of achieving a preprocessing step, in terms of the number of instances solved, and of the relative performance of the two preprocessors used (pmc and B + E), for each of the four model counters used downstream. Each cell of the table gives the number of wins corresponding to the preprocessing technique r associated with its row vs. the one c associated with its column. A win is an instance that has been solved by the model counter under consideration (given the time and memory limits considered in the experiments) when equipped with the preprocessing technique made precise by r but not solved when this model counter was equipped with c. The table shows, for instance, that Cachet equipped with B + E has been able to solve 80 instances that Cachet "alone" (i.e., when no preprocessing technique has been

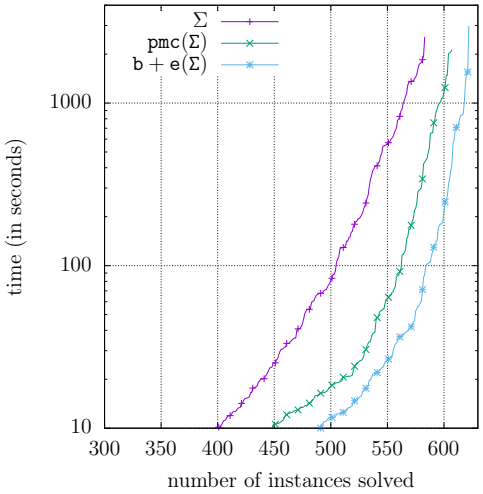


(a) Cachet

(b) SharpSAT



(c) C2D



(d) d4

Figure 1: Performances of Cachet, SharpSAT C2D and d4 depending on the preprocessing technique used (B + E, pmc, none).

applied first) was unable to solve (of course, for the same time-out and memory-out values, as made precise above). The other way around, Cachet "alone" has been able to solve 3 instances that Cachet equipped with B + E was unable to solve.

Cachet	no preproc.	pmc	B + E
no preproc.	0	6	3
pmc	39	0	3
B + E	80	47	0

SharpSAT	no preproc.	pmc	B + E
no preproc.	0	15	3
pmc	45	0	3
B + E	89	59	0

C2D	no preproc.	pmc	B + E
no preproc.	0	3	1
pmc	58	0	3
B + E	75	22	0

d4	no preproc.	pmc	B + E
no preproc.	0	4	3
pmc	27	0	4
B + E	41	19	0

Table 2: Comparison of the three preprocessing techniques ("no preprocessing technique", pmc, and B + E) for several model counters. Each cell of the table gives the number of wins corresponding to the preprocessing technique r associated with its row vs. the one c associated with its column. A win is an instance that has been solved by the model counter under consideration when equipped with the preprocessing technique made precise by r but not solved when this model counter was equipped with c.

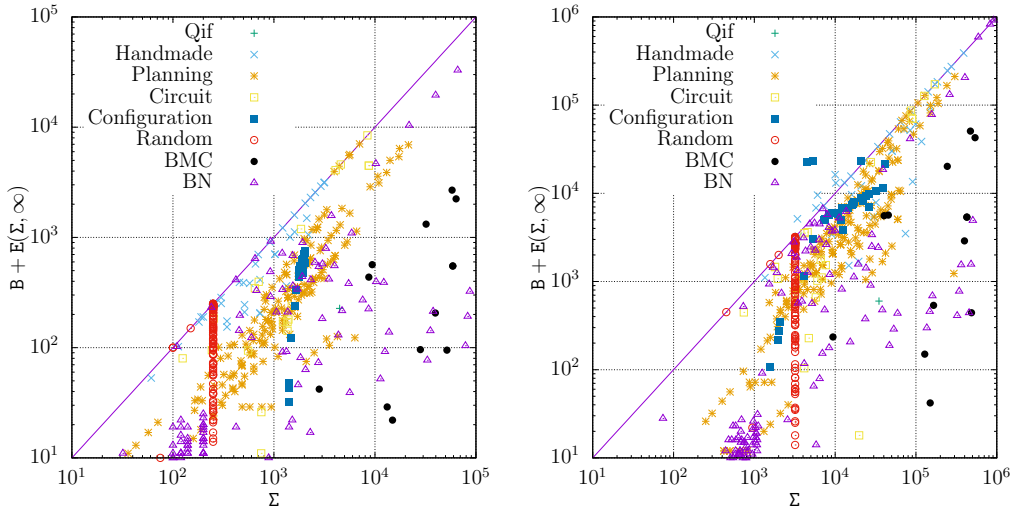
Clearly enough, the results reported in Table 2 show that adding a preprocessing step is in general useful, and that B + E typically performs better than pmc, even if those observations cannot be lifted to each instance taken separately. The fact that a preprocessing technique does not always help is easy to understand since, for instance, it can be the case that no reductions of the input instance are feasible (this is often the case when this instance has already been preprocessed). Thus, when no gates exist in the input CNF formula, the time used to search for them is just wasted (and this wasted time can be much higher when B + E is used, compared to pmc since, so to say, B + E has the potential to "find out" many more gates than the ones pmc can discover in the input).

B + E *vs. no preprocessing technique.* In order to determine how much applying B + E leads to reduce the input CNF formula Σ compared to the case when no preprocessing technique is used, we considered two measures for assessing the reduction of Σ: #var(Σ), the number of variables of Σ, and #lit(Σ), the number of literals occurring in Σ (i.e., the size of Σ).[10] The rationale for considering the number of variables and the number of literals of a CNF formula Σ is that the complexity of counting the number of models of Σ depends on both of them: the smaller the better. A significant decrease of #var(Σ) or #lit(Σ) may thus explain (but is not enough to guarantee) an improved performance of the subsequent model counting process.

Another measure that would also make sense to consider is the treewidth of the primal graph associated with Σ. Treewidth is a structural parameter that is widely used in the complexity analysis of graph algorithms. Especially, counting the number of models of a CNF formula Σ is fixed-parameter tractable when the parameter is the treewidth of the primal graph associated with Σ [57]. However, we refrained from computing this measure (while we did it in [32]), because computing its value is NP-hard and we did not find any satisfying piece of software for achieving such a computation in reasonable time given the sizes of the instances considered in our experiments, which can be huge. For instance, the instance comm_p10_p_t5 from the Planning family which has been solved by B + E+Cachet, by B + E+SharpSAT, and by B + E+C2D within the allocated resource bounds, but not by the other approaches, is over 8979 variables and contains 36995 clauses.

Empirically, the results are presented on the two scatter plots reported in Fig-

[10] We could consider the number #cl of clauses in the CNF formula as an alternative measure.



(a) #var reductions (b) #lit reductions

Figure 2: Reductions achieved by B + E in terms of number of variables and in terms of the size of the formula (comparison with the case when no preprocessing technique has been applied). Each point corresponds to an instance Σ, its x-coordinate corresponds to the value of the measure (#var or #lit) when no pre-processing is used, while its y-coordinate corresponds to the value of the same measure on B + E(Σ).

ure 9a, and Figure 9b. In these figures, each point corresponds to an instance Σ, its x-coordinate corresponds to the value of the measure (#var or #lit) when no pre-processing is used, while its y-coordinate corresponds to the value of the same measure on B + E(Σ) (with max#Conflicts = ∞). The scales used for both coordinates are logarithmic.

Clearly enough, using B + E often leads to large reductions for both measures. The benefits appear as very significant for instances from the Planning family, the BMC family, and, to some extent, for the BN family. As to #lit reduction, we can observe that for some benchmarks it is, so to say, "negative", that is, the size of the output CNF formula is sometimes slightly larger than the size of the input. This comes from the fact that E only ensures that the number of clauses does not increase whenever a variable is eliminated (which is not the same as guaranteeing that the number of literals will not increase).

We have also evaluated how much B + E leads to reduction of the overall model counting time compared to the case when no preprocessing technique is used. The results are presented on the four scatter plots (with logarithmic scales) reported in Figure 3a, Figure 3b, Figure 3c, and Figure 3d for the "direct" model

counters Cachet and SharpSAT and the compilation-based counters C2D and d4 considered downstream.

Each point corresponds to an instance Σ (a CNF formula), its x-coordinate corresponds to the time (in seconds) required to compute $\|\Sigma\|$ by calling the model counter on it, while its y-coordinate corresponds to the time required to compute $\|\Sigma\|$ by computing B + E(Σ) (with max#Conflicts= ∞) first, then calling the model counter on the resulting CNF formula. Again, whatever the downstream model counter, applying B + E appears as useful in many cases, when the overall model counting time can be significantly reduced. This is particularly the case for the model counter C2D, for which improvements are obtained very often (actually, for 624 instances over 703). This comes from the fact that the cutsets of the nodes of the decomposition tree of B + E(Σ) used by C2D for guiding the decomposition process and generating a Decision-DNNF representation are typically of smaller size than the cutsets of the nodes of the decomposition tree of Σ.

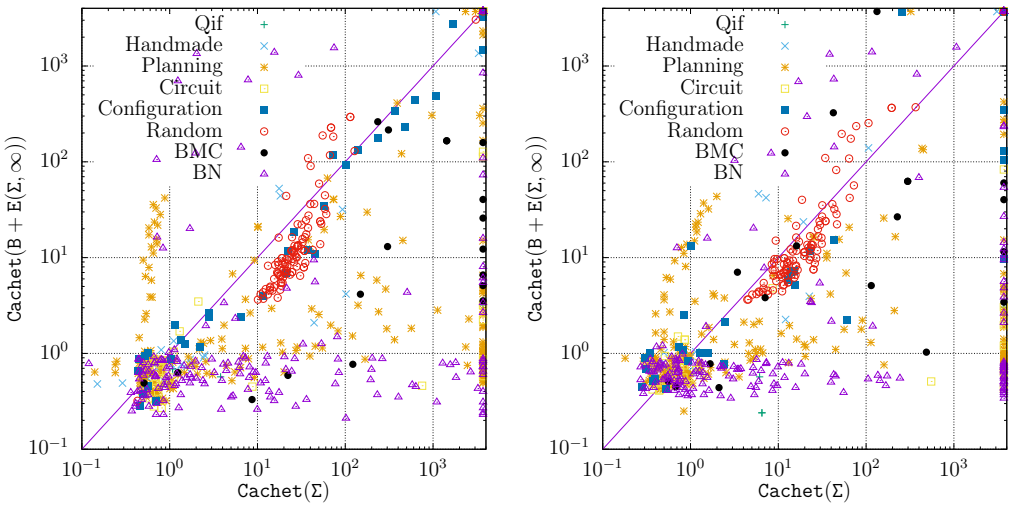
In our experiments, we have also measured the preprocessing times. Table 3 indicates for several time limits (in seconds) the number of benchmarks (out of 703) for which B + E has returned a CNF formula within the time limit.

time limit (in seconds)	number of instances preprocessed
≤ 1	436
≤ 5	465
≤ 10	558
≤ 100	647
≤ 1000	673
≤ 3600	683

Table 3: Number of instances preprocessed by B + E within a given amount of time.

More than 79% of the instances have been preprocessed in less than 10s, showing that B + E is quite efficient in practice. The instances that B + E was unable to preprocess within 3600s or those for which more than 100s have been spent do not belong to a specific family. The smallest instance that has not been preprocessed by B + E within 3600s contains "only" 7480 clauses. Conversely, B + E was able to preprocess very large instances within 100s (one of them contains 111682 clauses).

Figure 4 makes precise for each instance the preprocessing time used by B + E with respect to the total run time needed to "solve" it (i.e., to count its number of models) using C2D downstream (the compilation-based model counter that



(a) Cachet (b) SharpSAT

(c) C2D (d) d4

Figure 3: Performances of the model counters depending on the preprocessing technique used (B + E vs. no preprocessing technique). Each point corresponds to an instance Σ, its x-coordinate corresponds to the time (in seconds) required to compute $\|\Sigma\|$ by calling the model counter on it, while its y-coordinate corresponds to the time required to compute $\|\Sigma\|$ by computing B + E(Σ) first, then calling the model counter on the resulting CNF formula.

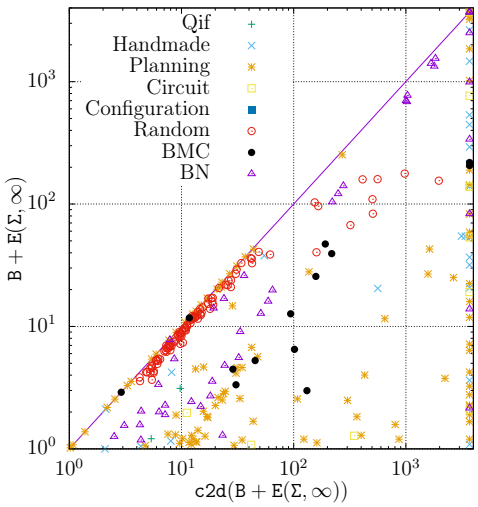
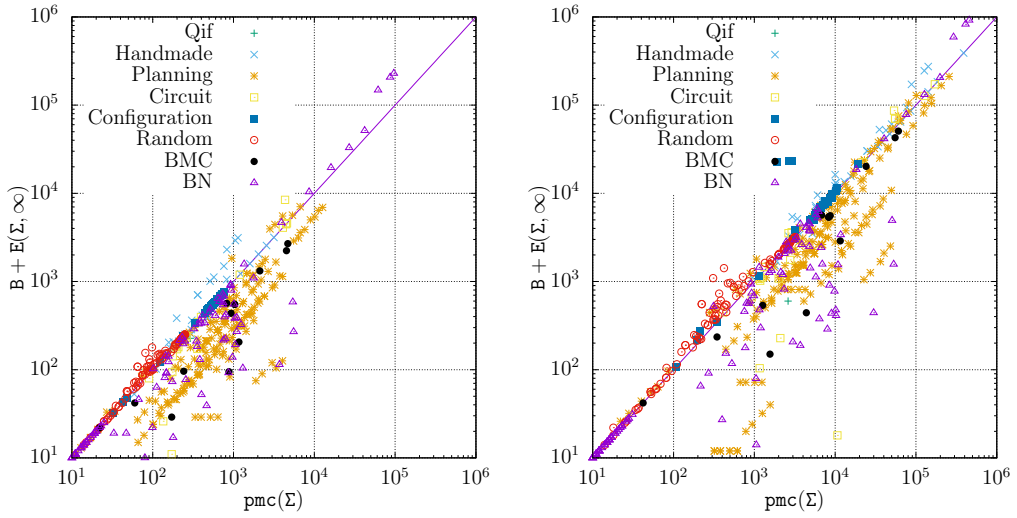


Figure 4: Preprocessing time with respect to the total run time needed to solve an instance when C2D is used downstream.

proved to be the most efficient counter in our experiments, in terms of the number of instances "solved", when it was equipped with B + E, see Table 1). Each point corresponds to an instance Σ (a CNF formula), its y-coordinate corresponds to the time (in seconds) required to compute B + E(Σ), while its x-coordinate corresponds to the time required to compute $\|$B + E(Σ)$\|$ using C2D.

One can observe on Figure 4 that the time required by C2D to count the number of models of its input is often significantly higher than the time used by B + E(Σ) for generating this input from Σ (this is quite salient for many instances from the Planning family). For other instances (especially those from the Random family), much of the total run time is spent by B + E, but one cannot conclude from it that using B + E for preprocessing those instances was a bad idea. Indeed, the preprocessing times can be (relatively) long – i.e., represent a large part of the overall computation times – just because B + E does almost all the job (it may happen that the simplification of the instance is so important that the downstream model counter has almost nothing to do afterwards).

B + E *vs.* pmc. In order to compare B + E with our previous preprocessor pmc, we computed similar measurements as in the previous paragraph, but this time, using pmc to preprocess the instances. Considering the values of the two measures #var(Σ) and #lit(Σ), we wanted first to compare the reductions achieved by B + E upstream with the ones achieved by pmc.



(a) #var reductions (b) #lit reductions

Figure 5: Reductions achieved by B + E in terms of number of variables and in terms of the size of the formula (comparison with pmc). Each point corresponds to an instance Σ, its x-coordinate corresponds to the value of the measure (#var or #lit) when pmc is used, while its y-coordinate corresponds to the value of the same measure on B + E(Σ).

The results are presented on the two scatter plots reported in Figure 5a, and Figure 5b. In these figures, each point corresponds to an instance Σ, its x-coordinate corresponds to the value of the measure (#var or #lit) when pmc is used, while its y-coordinate corresponds to the value of the same measure on B + E(Σ) (with max#Conflicts = ∞). The scales used for both coordinates are logarithmic.

Clearly enough, using B + E instead of pmc often leads to large reductions for both measures. The benefits appear as very significant for instances from the Planning family, the BMC family, and to some extent for the BN family. We can observe on Figure 5 that using pmc instead of B + E leads sometimes to remove more variables and/or more literals in the formula, but this does not happen very frequently, and typically, the reductions achieved in such cases are not so huge.

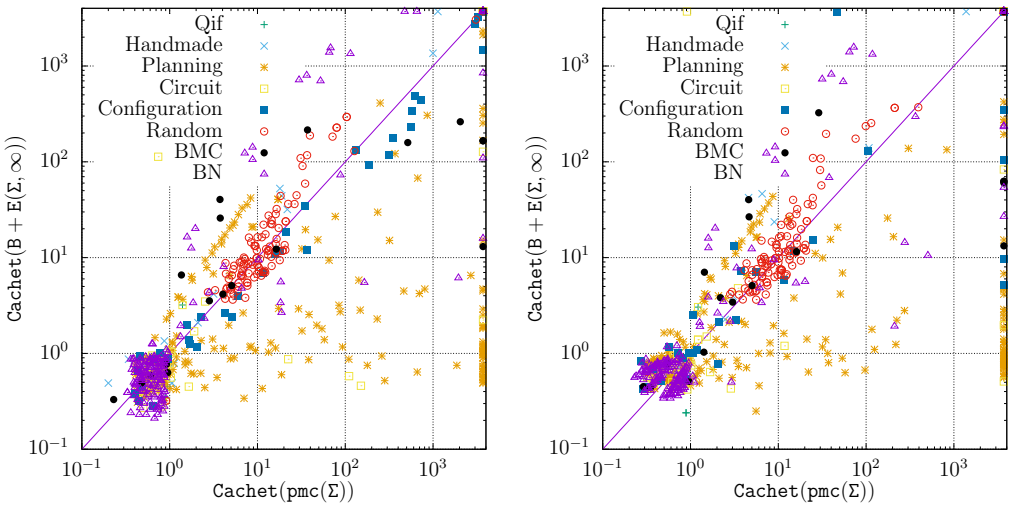
We have also evaluated how much B + E leads to reduce the overall model counting time compared to the case when pmc is used instead. The results are presented on the four scatter plots (with logarithmic scales) reported in Figure 6a, Figure 6b, Figure 6c and 6d for the "direct" model counters Cachet and SharpSAT and the compilation-based counters C2D and d4 considered downstream. Again, each point corresponds to an instance Σ (a CNF formula), its x-

coordinate corresponds to the time (in seconds) required to compute $\|\Sigma\|$ by calling first pmc on it and then the model counter on the resulting CNF formula, while its y-coordinate corresponds to the time required to compute $\|\Sigma\|$ by computing B + E(Σ) (with max#Conflicts= ∞) first, then calling the model counter on the resulting CNF formula.

We can observe that whatever the downstream model counter among the four ones we have considered, B + E appears typically as a better preprocessor than pmc in the sense that it leads typically to improved performances (smaller computation times). The rightmost parts of the two scatter plots cohere with the results reported in Table 1, showing a number of instances that can be solved by any of the model counters when B + E has been applied first, while they cannot be solved within the time limit of 1h when pmc is used instead. Of course, the computational benefits that are reported on these plots are less impressive than those offered by B + E compared to the case when no preprocessing technique is applied (pmc is quite a "good" preprocessor in many cases).

Detailed results on some instances. Finally, Tables 4 and 5 report some detailed results for a sample of the instances used in the experiments. In the two tables, the five leftmost columns characterize the instances used (by providing respectively, its family, the name of the instance, its number of variables #var, its number of clauses #cl, its size #lit in number of literals).

In Table 4, column "time B+E" gives the overall preprocessing time, which is the sum of the time "time B" needed to find a bipartition $\langle \mathrm{I}, \mathrm{O} \rangle$, plus the time "time E" required by the elimination. Columns "#input", "#output", and "#elim" give respectively the number of variables of I, of O, and the number of variables from O that have been forgotten or assigned during the elimination step. Finally, the last three columns indicate the number of variables #var, the number of clauses #cl, and the size #lit in number of literals of the output CNF formula (the reductions achieved can thus be computed by subtracting those values from the corresponding ones for the input CNF formula). It can be observed that the number of variables in the output CNF formula can be strictly lower than the number of variables in the input CNF formula Σ, minus the number of variables from O that have been forgotten or assigned during the elimination process. This comes from the fact that the elimination step can lead to remove some additional variables. As a matter of illustration, suppose that $\Sigma = (\neg \mathrm{a} \vee \mathrm{b}) \wedge (\mathrm{a} \vee \neg \mathrm{b})$ and that the bipartition $\langle \{\mathrm{a}\}, \{\mathrm{b}\} \rangle$ has been found. Eliminating b in Σ leads to remove the two clauses in Σ so that the output CNF formula is the empty conjunction of clauses. Hence, two variables have been removed, and not just one (while the cardinality of O is 1).



(a) Cachet (b) SharpSAT

(c) C2D (d) d4

Figure 6: Performances of the model counters depending on the preprocessing technique used (B + E vs. pmc). Eeach point corresponds to an instance Σ, its x-coordinate corresponds to the time (in seconds) required to compute $\|\Sigma\|$ by calling first pmc on it and then the model counter on the resulting CNF formula, while its y-coordinate corresponds to the time required to compute $\|\Sigma\|$ by computing B + E(Σ) first, then calling the model counter on the resulting CNF formula.

In Table 5, after the description of the input CNF instance Σ (five first columns), one can find three groups of columns, corresponding to the model counting task of Σ for three scenarios: no preprocessing technique (group Σ), use of pmc (group pmc(Σ)), and use of B + E (group B + E(Σ)). Within in each group, one can find the time (in seconds) needed by each one of the four counters Cachet, SharpSAT, C2D and d4 for making its job (it includes the preprocessing time, if any). "TO" means that a time out has been reached, and "MO" that the process has aborted due to insufficient memory. When a preprocessing technique took place (groups pmc(Σ) and B + E(Σ)), columns "time pmc" and "time B+E" indicate (respectively) the time spent by the application of the preprocessing technique. Columns #var, #cl, #lit indicate (respectively) the number of variables, number of clauses, and size of the CNF formula that has resulted from the application of the preprocessing technique and then be used as input of the model counters.

Benchmark Informations					B+E(Σ, ∞)								
Family	Name	#var	#cl	#lit	time B+E	time B	time E	#input	#output	#elim	#var	#cl	#lit
BN	50-10-9-q	460	720	1921	0.03	0.02	0.00	361	99	42	196	340	1261
BN	90-50-9-q	12300	19600	34149	16.86	16.85	0.00	9806	2494	1959	390	709	2426
BN	blockmap_05_01.net	1411	2737	5413	0.02	0.01	0.00	14	1397	1027	19	28	65
BN	blockmap_22_03.net	119003	247486	504902	57.15	36.09	21.05	261	118742	93182	585	1640	4915
BN	fs-01.net	32	38	74	0	0.00	0.00	13	19	9	11	12	28
BN	fs-19.net	65930	113924	260642	698.32	696.68	1.63	27949	37981	28025	32889	52573	131575
BN	mastermind_03_08_04.net	4720	10920	23608	0.46	0.44	0.01	216	4504	4119	216	180	376
BN	mastermind_10_08_03.net	5887	12901	28468	1.60	1.55	0.04	918	4969	4604	1093	1444	3383
BN	or-50-5-1-UC-10	100	250	653	0.00	0.00	0.00	42	58	47	4	1	4
BN	or-100-20-10-UC-10	200	500	1309	0.00	0.00	0.00	84	116	91	18	5	19
BN	sat-grid-pbl-0010	110	191	738	0.00	0.00	0.00	102	8	4	102	143	521
BN	sat-grid-pbl-0030	930	1771	7018	0.61	0.61	0.00	922	8	5	922	1697	6671
Config	C129_FR	1888	7404	23769	0.13	0.13	0.00	703	1185	21	554	2643	7998
Config	C140_FC	1828	5451	14572	0.10	0.10	0.00	605	1223	13	472	1793	5887
Config	C140_FV	1843	8056	33705	0.12	0.12	0.00	671	1172	17	539	2173	6948
Handmade	5_100_sd_schur	500	26860	78510	289.65	289.63	0.01	388	112	98	388	11669	52832
Handmade	ais6	61	581	1351	0.00	0.00	0.00	12	49	8	53	361	1105
Handmade	fphp-015-020	300	4965	10200	0.01	0.01	0.00	285	15	15	285	4665	13110
Handmade	lang12	576	13584	28134	42.59	42.55	0.04	134	442	102	384	4872	12142
Handmade	lang23	2116	96370	196489	2522.80	2522.13	0.67	633	1483	322	1495	38141	93997
Handmade	ls10-normalized	657	4761	10017	2277.82	2277.81	0.00	440	217	81	576	4680	16290
Planning	blocks_right_2_p_t5	406	1901	5407	0.02	0.01	0.00	65	341	299	107	678	2210
Planning	blocks_right_6_p_t6	3337	30517	82699	58.59	58.11	0.48	545	2792	1554	1784	22512	65591
Planning	bomb_b5_t1_p_t5	564	1086	2796	0.01	0.01	0.00	228	336	300	0	0	0
Planning	bomb_b20_t5_p_t10	14100	28025	75525	34.27	34.07	0.20	6575	7525	7250	0	0	0
Planning	coins_p01_p_t5	872	2353	5901	0.06	0.06	0.00	303	569	532	176	560	1256
Planning	coins_p10_p_t10	3898	9879	24559	1.43	1.42	0.01	1505	2393	2294	634	2443	5569
Planning	comm_p10_p_t10	17539	75516	191696	25.44	24.85	0.59	3853	13686	10353	6402	37147	104679
Planning	emptyroom_d4_g2_p_t5	188	584	1532	0.00	0.00	0.00	28	160	152	44	168	576
Planning	hanoi5	1931	14468	35856	0.00	0.00	0.00	0	1931	16	0	0	0
Planning	logistics.a	828	6718	17915	0.05	0.04	0.00	153	675	489	265	1078	3129
Planning	medium	116	953	2133	0	0.00	0.00	1	115	35	0	0	0
Planning	prob001.pddl	939	3785	7727	0.02	0.02	0.00	183	756	168	209	444	1061
Planning	prob012.pddl	2324	31857	64476	0.51	0.49	0.01	681	1643	701	850	4681	10551
Planning	uts_k10_p_t6	13047	89563	215484	39.94	39.41	0.53	2580	10467	6366	3880	54591	121740
Planning	uts_k10_p_t10	21451	148891	358620	2281.53	2280.69	0.84	4260	17191	10130	6920	93351	211340
BMC	bmc-galileo-9	63624	326999	852078	22.72	19.85	2.87	169	63455	30641	2238	11032	42720
BMC	bmc-ibm-1	9685	55870	149914	1.91	1.87	0.03	1033	8652	2809	392	1002	3616
BMC	bmc-ibm-13	13215	65728	174164	0.32	0.31	0.01	145	13070	2986	29	30	150
BMC	cnt06.shuffled	762	2469	6753	0.00	0.00	0.00	0	762	5	0	0	0
BMC	cnt10.shuffled	20470	68561	187229	0.02	0.02	0.00	0	20470	17	0	0	0
Circuit	3bitadd_32	8704	32316	89472	158.56	158.56	0.00	8704	0	0	4480	26475	70341
Circuit	alu2_gr_rcs_w8.shuffled	4080	83902	170864	86.22	86.20	0.02	4064	16	2	4078	83900	172438
Circuit	c432.isc	196	514	1258	0.00	0.00	0.00	36	160	160	0	0	0
Circuit	c880.isc	417	1060	2466	0.00	0.00	0.00	60	357	357	0	0	0
Circuit	c880_gr_rcs_w7.shuffled	4592	61745	126770	23.26	23.26	0.00	4585	7	1	4591	61744	127153
Circuit	c1355.isc	555	1546	3610	0.06	0.06	0.00	41	514	461	94	304	1080
Circuit	c7552.isc	3185	8588	19808	0.29	0.28	0.00	208	2977	2977	5	5	18
Circuit	ssa7552-038	1501	3575	11823	0.07	0.06	0.00	179	1322	1055	239	475	1536
Circuit	ssa7552-160	1391	3126	10151	0.05	0.04	0.00	133	1258	1017	180	330	1112
QIF	10random	587	1685	4157	0.01	0.01	0.00	129	458	379	0	0	0
QIF	binsearch.32	4473	14011	35245	1.64	1.42	0.21	391	4082	3788	227	87	600
QIF	binsearch.32.pp	251	1917	7304	0.00	0.00	0.00	32	219	218	0	0	0
QIF	mixdup	309	452	1092	0.00	0.00	0.00	161	148	76	0	0	0
QIF	sum.32	639	1708	4356	0.02	0.02	0.00	129	510	438	0	0	0
Random	uf250-01	250	1065	3195	168.94	168.94	0.00	249	1	0	250	1065	3195
Random	uf250-0100	250	1065	3195	0.21	0.21	0.00	103	147	93	143	338	868
Random	wff.3.75.315	75	315	945	0	0.00	0.00	10	65	55	10	11	22
Random	wff.4.100.500	100	500	2000	0.00	0.00	0.00	100	0	0	100	500	2000

Table 4: Effectiveness of `B` + `E` in terms of preprocessing time and of reduction achieved — some instances.

Benchmark Informations					Σ				pmc(Σ)								B + E(Σ)							
Family	Name	#var	#cl	#lit	Cachet	SharpSAT	c2d	d4	time pmc	#var	#cl	#lit	Cachet	SharpSAT	c2d	d4	time B+E	#var	#cl	#lit	Cachet	SharpSAT	c2d	d4
BN	50-10-9-q	460	720	1921	TO	1149.58	2.38	2.52	0.01	239	412	1508	65.54	28.28	1.49	1.10	0.03	196	340	1261	64.43	20.24	1.06	0.67
BN	90-50-9-q	12300	19600	34149	TO	MO	TO	1848.38	8.05	1094	2117	6761	TO	MO	20.08	34.03	16.86	390	709	2426	40.49	27.47	34.88	26.04
BN	blockmap_05_01.net	1411	2737	5413	0.46	0.20	12.72	0.24	0.03	33	92	247	0.72	0.47	5.28	0.41	0.02	19	28	65	0.58	0.98	5.16	0.38
BN	blockmap_22_03.net	119003	247486	504902	74.13	1075.92	1370.26	338.56	81.50	5413	18167	47548	86.43	90.36	490.64	87.03	57.15	585	1640	4915	122.17	117.76	573.83	1553.54
BN	fs-01.net	32	38	74	0.49	0.44	0.66	0.24	0.00	11	12	28	0.73	0.45	0.74	0.42	0	11	12	28	0.53	0.93	0.61	0.25
BN	fs-19.net	65930	113924	260642	TO	MO	TO	TO	403.49	26950	52934	127148	TO	MO	TO	MO	698.32	32889	52573	13157	TO	MO	TO	TO
BN	mastermind_03_08_04.net	4720	10920	23608	3.75	MO	53.45	9.30	0.24	750	2533	6799	3.82	1.50	22.43	2.71	0.46	216	180	376	1.10	1.57	17.37	3.43
BN	mastermind_10_08_03.net	5887	12901	28468	45.26	MO	105.30	2555.43	0.86	1776	4497	11791	19.26	MO	43.79	46.45	1.60	1093	1444	3383	2.76	3.08	30.06	5.51
BN	or-50-5-1-UC-10	100	250	653	0.57	0.51	1.55	0.52	0.00	4	1	4	0.64	0.45	0.91	0.43	0.00	4	1	4	0.51	0.02	0.54	0.19
BN	or-100-20-10-UC-10	200	500	1309	TO	MO	MO	MO	0.00	18	5	19	0.70	0.46	0.74	0.46	0.00	18	5	19	0.99	0.06	0.50	0.39
BN	sat-grid-pbl-0010	110	191	738	0.62	0.50	1.67	0.27	0.00	102	71	188	0.71	0.40	1.39	0.48	0.00	102	143	521	0.63	0.99	1.61	0.14
BN	sat-grid-pbl-0030	930	1771	7018	TO	MO	MO	651.30	0.11	922	669	1867	TO	MO	4.45	27.70	0.61	922	1697	6671	TO	MO	MO	3.42
Config	C129_FR	1888	7404	23769	71.49	42.46	44.72	109.55	0.08	555	2711	8076	308.27	MO	122.80	17.62	0.13	554	2643	7998	162.90	11.58	23.82	12.21
Config	C140_FC	1828	5451	14572	11.69	2.59	33.20	4.97	0.05	469	1781	5804	5.87	1.92	43.14	10.72	0.10	472	1793	5887	4.60	2.52	14.12	4.52
Config	C140_FV	1843	8056	33705	21.94	22.67	TO	160.04	0.09	540	2115	6491	18.58	15.95	16.37	16.97	0.12	539	2173	6948	12.59	14.97	30.50	7.57
Handmade	5_100_sd_schur	500	26860	78510	TO	MO	TO	MO	0.79	486	12155	35263	TO	MO	TO	MO	289.65	388	11669	52832	TO	MO	TO	TO
Handmade	ais6	61	581	1351	0.56	0.65	0.98	0.33	0.00	59	390	935	0.50	0.58	0.99	0.55	0.00	53	361	1105	0.59	0.17	0.99	0.40
Handmade	fphp-015-020	300	4965	10200	TO	MO	TO	TO	0.08	300	4965	10200	TO	MO	TO	TO	0.01	285	4665	13110	TO	MO	MO	MO
Handmade	lang12	576	13584	28134	3300.24	3061.78	TO	210.83	0.11	198	4724	9970	947.49	1272.12	1360.22	48.28	42.59	384	4872	12142	1999.86	TO	1269.40	185.92
Handmade	lang23	2116	96370	196489	TO	TO	MO	TO	1.93	759	37515	77169	TO	TO	TO	TO	2522.80	1495	38141	93997	TO	TO	TO	TO
Handmade	ls10-normalized	657	4761	10017	TO	TO	TO	TO	0.04	657	4761	10017	TO	TO	TO	TO	2277.82	576	4680	16290	TO	TO	TO	TO
Planning	blocks_right_2_p_t5	406	1901	5407	0.71	0.37	5.86	0.43	0.02	189	1078	2966	0.58	0.59	4.05	0.64	0.02	107	678	2210	0.65	1.08	3.48	0.43
Planning	blocks_right_6_p_t6	3337	30517	82699	TO	MO	TO	TO	1.52	2467	24643	61245	TO	MO	TO	TO	58.59	1784	22512	65591	TO	MO	TO	MO
Planning	bomb_b5_t1_p_t5	564	1086	2796	1.00	0.30	9.33	0.34	0.01	180	208	590	0.93	0.54	2.03	0.52	0.01	0	0	0	0.91	0.11	1.67	0.28
Planning	bomb_b20_t5_p_t10	14100	28025	75525	0.92	1.23	228.69	2.37	5.49	6750	7100	20450	6.10	5.85	86.52	5.49	34.27	0	0	0	16.79	46.36	40.40	21.62
Planning	coins_p01_p_t5	872	2353	5901	TO	MO	13.55	35.51	0.10	390	1173	3069	TO	MO	7.11	19.30	0.06	176	560	1256	0.58	1.07	6.21	0.29
Planning	coins_p10_p_t10	3898	9879	24559	TO	MO	MO	MO	0.48	2051	6556	16491	TO	MO	MO	MO	1.43	634	2443	5569	2.48	1.82	20.06	2.59
Planning	comm_p10_p_t10	17539	75516	191696	TO	MO	MO	TO	5.90	11209	61138	156145	TO	MO	TO	MO	25.44	6402	37147	10467	TO	MO	MO	MO
Planning	emptyroom_d4_g2_p_t5	188	584	1532	0.69	0.28	2.95	0.35	0.00	108	323	892	0.52	0.52	1.82	0.51	0.00	44	168	576	0.52	0.04	1.30	0.31
Planning	hanoi5	1931	14468	35856	TO	TO	TO	0.88	1.46	0	0	0	1.63	3.39	1.55	1.70	0.00	0	0	0	1.90	0.58	6.95	0.64
Planning	logistics.a	828	6718	17915	2.64	0.41	TO	10.15	0.05	285	1082	2805	1.08	0.61	6.16	0.80	0.05	265	1078	3129	1.53	0.19	5.62	0.82
Planning	medium	116	953	2133	0.57	0.30	1.72	0.27	0.00	0	0	0	0.83	0.46	0.24	0.46	0	0	0	0	0.72	0.40	1.01	0.24
Planning	prob001.pddl	939	3785	7727	0.67	0.61	12.11	0.31	0.01	255	531	1128	0.52	0.51	5.70	0.46	0.02	209	444	1061	0.78	1.03	4.23	0.19
Planning	prob012.pddl	2324	31857	64476	57.44	8.92	1453.89	681.12	0.44	1018	5576	11494	155.10	5.33	184.59	117.48	0.51	850	4681	10551	75.49	10.20	284.51	314.93
Planning	uts_k10_p_t6	13047	89563	215484	TO	MO	TO	MO	4.93	7627	68520	153437	TO	MO	TO	MO	39.94	3880	54591	12174	TO	MO	TO	MO
Planning	uts_k10_p_t10	21451	148891	358620	TO	MO	TO	MO	10.88	12671	114438	256483	TO	MO	TO	MO	2281.53	6920	93351	21134	TO	MO	TO	TO
BMC	bmc-galileo-9	63624	326999	852078	313.56	42.24	TO	203.68	87.89	4524	93463	260369	140.41	93.33	TO	1179.31	22.72	2238	11032	42720	41.00	43.61	TO	215.41
BMC	bmc-ibm-1	9685	55870	149914	15.40	2.14	260.64	7.55	1.24	1242	2855	7370	2.19	2.21	33.62	1.86	1.91	392	1002	3616	3.11	3.67	27.07	4.50
BMC	bmc-ibm-13	13215	65728	174164	TO	114.12	TO	8.83	2.77	172	745	2105	5.34	5.20	43.09	5.14	0.32	29	30	150	4.67	5.19	47.80	5.25
BMC	cnt06.shuffled	762	2469	6753	1.34	0.20	11.64	0.39	0.01	0	0	0	0.67	0.46	0.81	0.42	0.00	0	0	0	0.48	1.07	2.54	0.33
BMC	cnt10.shuffled	20470	68561	187229	TO	MO	TO	20.10	2.63	0	0	0	17.71	16.05	16.39	16.63	0.02	0	0	0	16.76	17.15	87.61	11.69
Circuit	3bitadd_32	8704	32316	89472	TO	MO	TO	TO	19.61	4480	24323	55214	TO	TO	TO	MO	158.56	4480	26475	70341	TO	MO	TO	TO
Circuit	alu2_gr_rcs_w8.shuffled	4080	83902	170864	TO	MO	TO	TO	1.29	4080	83902	170864	TO	MO	TO	TO	86.22	4078	83900	17243	TO	MO	TO	TO
Circuit	c432.isc	196	514	1258	0.73	0.68	2.50	0.31	0.00	93	266	733	0.53	0.55	1.69	0.43	0.00	0	0	0	0.63	1.09	1.32	0.15
Circuit	c880.isc	417	1060	2466	TO	MO	25.79	242.89	0.01	142	404	1141	15.97	4.32	2.84	2.00	0.00	0	0	0	0.57	1.04	1.48	0.14
Circuit	c880_gr_rcs_w7.shuffled	4592	61745	126770	TO	MO	TO	TO	1.16	4592	61745	126770	TO	MO	TO	TO	23.26	4591	61744	12715	TO	MO	TO	TO
Circuit	c1355.isc	555	1546	3610	TO	MO	11.61	162.36	0.03	175	854	2297	TO	MO	18.93	124.67	0.06	94	304	1080	60.66	82.50	1.96	0.91
Circuit	c7552.isc	3185	8588	19808	TO	MO	1014.39	TO	0.26	819	3158	10113	TO	MO	50.87	MO	0.29	5	5	18	1.19	1.41	10.63	1.66
Circuit	ssa7552-038	1501	3575	11823	0.72	0.40	14.06	0.34	0.03	263	532	1548	0.57	0.59	5.05	0.57	0.07	239	475	1536	1.29	0.15	5.41	0.40
Circuit	ssa7552-160	1391	3126	10151	0.63	0.34	12.66	0.32	0.02	208	413	1236	0.31	0.99	5.27	0.50	0.05	180	330	1112	0.80	0.75	4.59	0.31
QIF	10random	587	1685	4157	0.60	0.61	6.40	0.58	0.03	41	301	883	0.61	0.35	1.81	0.61	0.01	0	0	0	0.17	0.81	1.87	0.57
QIF	binsearch.32	4473	14011	35245	TO	MO	TO	MO	0.55	292	1022	2710	2.14	1.17	10.94	1.10	1.64	227	87	600	2.01	3.11	14.43	21.46
QIF	binsearch.32.pp	251	1917	7304	0.82	6.07	25.15	0.68	0.04	65	595	1606	0.49	0.60	1.59	0.58	0.00	0	0	0	0.08	0.21	1.30	0.54
QIF	mixdup	309	452	1092	0.59	0.53	2.07	0.52	0.00	0	0	0	0.57	0.59	0.11	0.51	0.00	0	0	0	0.69	0.59	0.59	0.30
QIF	sum.32	639	1708	4356	3.92	MO	MO	148.64	0.01	213	714	2386	TO	MO	4.96	1.03	0.02	0	0	0	0.24	0.83	1.81	0.61
Random	uf250-01	250	1065	3195	194.54	106.22	1746.68	21.79	13.92	250	1065	3195	83.07	117.83	1739.28	30.50	168.94	250	1065	3195	272.81	302.32	1950.96	176.48
Random	uf250-0100	250	1065	3195	34.10	24.74	119.89	7.75	8.34	124	218	501	10.05	9.82	10.94	9.91	0.21	143	338	868	10.67	10.78	12.58	7.43
Random	wff.3.75.315	75	315	945	0.59	0.21	1.39	0.34	0.00	9	9	18	0.38	0.45	1.11	0.38	0	10	11	22	0.52	0.62	0.92	0.30
Random	wff.4.100.500	100	500	2000	TO	MO	MO	MO	0.00	100	500	2000	TO	MO	MO	MO	0.00	100	500	2000	TO	MO	MO	MO

Table 5: Effectiveness of B + E in terms of the overall model counting task — some instances.

The results reported in Table 4 and Table 5 confirm the observations already made in light of the previous tables and figures. Mainly, using a preprocessing technique can be useful in terms of the simplification of the input instance and of the overall CPU time needed to achieve its model counting task (whatever the model counter), but this is not always the case. And B + E often leads to better performance than pmc, but, again, this is not always the case. More specifically, the two tables show that the reduction of the input CNF instance achieved by the preprocessing step (in terms of number of variables, number of clauses or size) can be very small (even null) but is quite large in some cases. When the reduction is large, the time needed to count models can be significantly lowered, but it increases for some instances. The time spent by B can be high even if the corresponding number of variables in the set O of output variables found is small. The time spent by E is typically small (but it is not the case when many variables are to be eliminated). The time required by B + E can be much higher than the one used by pmc, but it is not always the case.

5. Definability and Approximate Compilation

5.1. Definability and Exact Compilation

While B + E can be considered upstream to a Decision-DNNF compiler, like C2D or d4, when used as a model counter (as we explained it in Section 4), we cannot take advantage of B + Eto boost such a compiler when used for equivalence-preserving compilation. Indeed, B + E is not an equivalence-preserving preprocessing technique. Therefore, whenever B + E(Σ) is not equivalent to Σ, the Decision-DNNF representation of B + E(Σ) computed by the compiler will not be equivalent to Σ. Especially, while it is possible to count efficiently the number of models of a Decision-DNNF representation (and, more generally, of a d−DNNF representation) of Σ conditioned by any consistent term over *Var*(Σ), this possibility is lost if a Decision-DNNF representation of B + E(Σ) is computed instead. Indeed, in such a case, one cannot take account anymore of any conditioning of the variables that have been forgotten by B + E.

Example 6 (Example 4 cont'ed). *As a matter of illustration, consider again our running example. A Decision-*DNNF *representation of* B + E(Σ) = (a $\vee$ b) $\wedge$ (a $\vee$ c) *is given by* a $\wedge$ *ite*(b, c, $\top$)*, represented on Figure 7 (the node labelled by* b *is a decision node, i.e., an if ... then ... else ... (ite) node, where ite is a ternary connective whose semantics is given by ite*(x, y, z) = ($\neg$x$\wedge$y)$\vee$(x$\wedge$z)*. Conditioning this representation by* d $\wedge$ e *does not change it, and the number of models of it is*

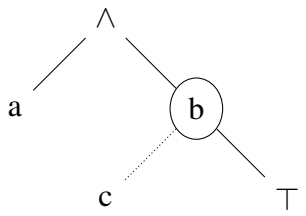


Figure 7: A Decision-DNNF representation equivalent to $\mathtt{B} + \mathtt{E}(\Sigma)$.

equal to 3. *Contrastingly, conditioning* Σ *by* $\mathrm{d} \wedge \mathrm{e}$ *leads to a formula equivalent to* $\mathrm{a} \wedge \mathrm{b} \wedge \mathrm{c}$*, thus having a single model.*

In order to address this issue, a (tentative) approach would be to keep in addition to a Decision-DNNF representation Σ^* of $\mathtt{B} + \mathtt{E}(\Sigma)$ all the gates Φ_I^x of the variables $x \in O$ on I in Σ that have been forgotten by E. Indeed, if $\langle I, O \rangle$ is a definability bipartition of Σ and E a subset of O such that for every $x \in E$, Φ_I^x is a gate of x on I in Σ, then $(\exists E.\Sigma) \wedge (\bigwedge_{x \in E}(x \leftrightarrow \Phi_I^x))$ is equivalent to Σ (hence, no information would be lost). The fact that Σ implies $(\exists E.\Sigma) \wedge (\bigwedge_{x \in E}(x \leftrightarrow \Phi_I^x))$ is obvious. The other way around, let us consider a model ω of $\exists E.\Sigma$ over $Var(\Sigma)$. Then there necessarily exists a model ω' of Σ over $Var(\Sigma)$ such that $\forall x_i \in Var(\Sigma) \setminus E$, $\omega(x_i) = \omega'(x_i)$ (see Corollary 5 in [42]). Since every variable from I belongs to $Var(\Sigma) \setminus E$, for each $x \in E$, ω is a model of Φ_I^x if and only if ω' is a model of Φ_I^x. Since ω is a model of $\bigwedge_{x \in E}(x \leftrightarrow \Phi_I^x)$, we must also have that $\omega(x) = \omega'(x)$ for each $x \in E$. Stated otherwise, $\omega = \omega'$, hence ω is a model of Σ.

It turns out that this approach is not satisfying, for two reasons. On the one hand, the computation of the gates Φ_I^x can be very demanding: the computation time can be huge, and above all, the sizes of the gates Φ_I^x can be huge as well (i.e., not polynomially bounded in $|\Sigma| + \#(I)$ [33]) which is problematic, even in the case the computation time could be neglected because it is an off-line compilation time. On the other hand, the resulting pair consisting of a Decision-DNNF representation Σ^* of $\exists E.\Sigma$ and a conjunction $\bigwedge_{x \in E}(x \leftrightarrow \Phi_I^x)\rangle$ of gates (one per variable in E) cannot be considered as a valuable compiled form of Σ: in the general case, it is NP-hard to determine the satisfiability of a conjunction of a Decision-DNNF formula α with a conjunction κ of equivalences $x \leftrightarrow \Phi_I^x$ with $x \notin Var(\alpha)$ and $Var(\Phi_I^x) \subseteq Var(\alpha)$, after its conditioning by a consistent term γ. Hardness is still the case when the gates Φ_I^x have a simple structure and are of small size (e.g., clauses or terms containing at most 3 literals). This can be easily shown via the following reduction from SAT. Let $\beta = \bigwedge_{i=1}^{m} \delta_i$ be a CNF

formula over $x_1, \ldots, x_n$. Let $\Sigma = \bigwedge_{i=1}^{m}(y_i \Leftrightarrow \delta_i)$ where $y_1, \ldots, y_m$ are fresh variables, distinct from $x_1, \ldots, x_n$. Clearly enough, $\langle\{x_1, \ldots, x_n\}, \{y_1, \ldots, y_m\}\rangle$ is a definability bipartition of Σ. Take $E = O$. The formula $\exists E.\Sigma$ is valid, which can be represented by a Decision-DNNF formula α consisting of the (decomposable) conjunction of n Decision-DNNF representations of valid clauses $x_i \vee \neg x_i$ ($i \in \{1, \ldots, n\}$). Obviously enough, we have $\Sigma \equiv \alpha \wedge \Sigma$. Consider now the consistent term $\gamma = \bigwedge_{i=1}^{m} y_i$. By construction, β is equivalent to $\Sigma|\gamma$, hence β is satisfiable if and only if $\Sigma|\gamma$ is satisfiable, and this completes the proof.

5.2. *From Exact Compilation to Approximate Compilation: The Case of Controllable Variables*

Interestingly, it makes sense to exploit the "Bipartition, then Eliminate" approach at work in B + E in the case when the set of variables of Σ can be partitioned into a set C of controllable variables (those that may be conditioned at some point for solving the problem under consideration and must be protected for this reason) and the remaining set $Var(\Sigma) \setminus C$ of variables (e.g., some auxiliary variables introduced for the encoding purpose). Indeed, in such a case, one can identify a simple condition under which B + E(Σ) is *query-equivalent* to Σ over C [58], meaning that

$$\exists(Var(\Sigma) \setminus C).B + E(\Sigma) \equiv \exists(Var(\Sigma) \setminus C).\Sigma$$

or equivalently, that for every formula φ such that $Var(\varphi) \subseteq C$, we have $\Sigma \models \varphi$ if and only if B + E(Σ) $\models \varphi$. This condition is made precise in Proposition 5.

Proposition 5. *Let Σ be a* CNF *formula and* C *be a finite subset of $\mathcal{P}$. If the bipartition $\langle I, O\rangle$ of Σ computed by* B *is such that* $C \subseteq I$*, then* B + E(Σ) *is query-equivalent to Σ over* C*.*

Proof: By definition of forgetting, $\exists(Var(\Sigma) \setminus C).\Sigma$ is query-equivalent to Σ over C. By construction, B + E returns a CNF formula equivalent to $\exists E.\Sigma$ where $E \subseteq O$. Hence, we have $\exists(Var(\Sigma)\setminus C).B + E(\Sigma) \equiv \exists(Var(\Sigma)\setminus C).(\exists E.\Sigma)$. When $C \subseteq I$, we have $C \cap O = \emptyset$ since $\langle I, O\rangle$ is a bipartition of Σ. Hence, $E \subseteq (Var(\Sigma)\setminus C)$, showing that $\exists(Var(\Sigma) \setminus C).(\exists E.\Sigma)$ is equivalent to $\exists(Var(\Sigma) \setminus C).\Sigma$, and this completes the proof. □

Ensuring that the condition $C \subseteq I$ is satisfied is not very demanding: it is enough to slightly modify B in such a way that the variables of the backbone of Σ that belongs to C are not put into O at line 1, that I is initialized with C at line 3, and that x takes its values in $V \setminus C$ at line 4. When the condition $C \subseteq I$ is satisfied,

compiling $\exists(\mathit{Var}(\Sigma) \setminus C).\mathtt{B} + \mathtt{E}(\Sigma)$ leads to an *approximate compilation* [59] of Σ on C, where all (but only) the consequences of Σ over C are guaranteed to be preserved.

The significance of Proposition 5 comes from that fact that in a number of AI applications such a set C of controllable variables can be pointed out. For instance, in model-based diagnosis, each variable of the propositional description of the system to be diagnosed can be forgotten when it does not correspond to an observation of the system and it does not encode the fact that a component of the system is faulty (or dually, that it works correctly), see e.g., [55].

5.3. Application to Planning

Another application area for which Proposition 5 can be useful is planning.

The classical planning setting. In classical planning, inputs take the form of tuples $P = (F, A, S, G)$, where F is a finite set of fluents, A is a set of deterministic actions with possibly conditional effects, S is a complete truth assignment of initial fluents in F (a description of the state of the system at start), and G is a partial assignment of final fluents in F representing the goal situation. A *plan* π for P is a sequence π of sets of actions, one per time point between 0 and $N-1$, that maps the initial state S to a goal state (i.e., a model of G). In order to solve P, we can encode it into a corresponding CNF formula Σ_P over the set of variables $(\bigcup_{i=0}^{N}\{f_i \mid f \in F\}) \cup \{a_i \mid a \in A, i = 0, \ldots, N-1\}$. Σ_P can be viewed as a compact representation of the transition model associated with A. In this encoding, f_i is true if and only if fluent f holds at time point i, and a_i is true if and only if action a holds at time point i. Since only deterministic actions are considered in A, the truth value of every fluent f_i ($f \in F, i \in 1, \ldots, N$) is fully determined in Σ_P as soon as the truth values of the variables $\{f_0 \mid f \in F\} \cup \bigcup_{i=0}^{N-1}\{a_i \mid a \in A\}$ are fixed (i.e., as soon as the initial state and the plan under consideration are specified). Focusing on $C = \{f_0 \mid f \in F\} \cup \{f_N \mid f \in F\} \cup \{a_i \mid a \in A, i = 0, \ldots, N-1\}$ instead of its superset $\mathit{Var}(\Sigma_P)$ is enough for solving efficiently a number of issues of interest, once an approximate compiled form consisting of a Decision-DNNF representation of $\mathtt{B} + \mathtt{E}(\Sigma_P)$ has been computed. For instance, we can determine in polynomial time whether a plan π exists for any S and G given on-line, we can enumerate such plans with polynomial delay, and we can also count in polynomial time how many π exist.

Some experiments. In order to determine the extent to which using $\mathtt{B} + \mathtt{E}$ on Σ_P before compiling it into a Decision-DNNF can be useful, we performed a number

of experiments, focusing on the 530 planning instances used in [60] and available on `www.cril.fr/KC/`. Those instances cover a range of different planning benchmarks, with varying horizon length. "blocks-n" refers to the famous blocks-world domain with n blocks. "bomb-m-n" is another popular domain involving m bombs, n toilets, and 2 actions ("dunking" a bomb into a toilet, and "flushing" a clogged toilet). "comm-m-n" is an IPC5 problem about communication signals with m stages, n packets, and 5 actions. "emptyroom-n" is about navigating a robot in an n $\times$ n empty grid. Finally, "safe-n" is about opening a safe with n possible combinations.

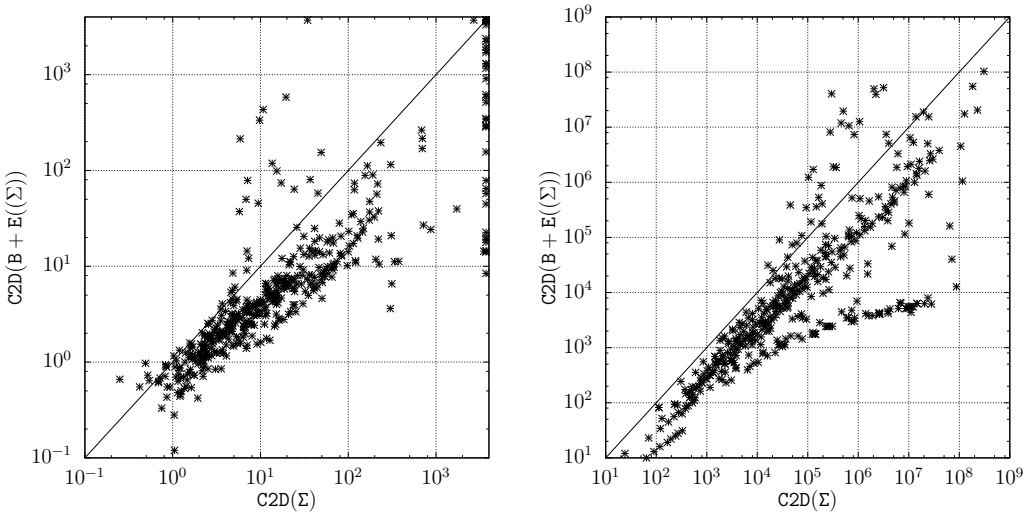
All instances described in PDDL were translated into `CNF` theories using the DIMACS format, and then compiled using `C2D` and/or `d4`. For each instance, the set of controllable variables C consist of the variables corresponding to the actions in A (whatever the instant when they occur), the variables corresponding to the initial state S (all the fluents at time 0), and those corresponding to the goal (all the fluents at time N). `B` has been modified to ensure that C $\subseteq$ I in the resulting bipartition, as explained above (which is mandatory to guarantee that no variable in C has been eliminated).

Figure 8 (a–b–c–d) contains scatter plots showing the impact of using `B` + `E` as a preprocessor before compiling the encodings Σ_P of the instances into Decision-`DNNF` representations. Each point corresponds to a planning instance. The two compilers `C2D` (a–b) and `d4` (c–d) have been considered. Both the compilation times (a–c) in seconds and the sizes of the compiled forms (b–d) in number of edges have been measured. As usual, the times reported include the preprocessing times if any.

The scatter plots in Figure 8 show that in general some computational benefits are obtained when using `B` + `E`. They concern both the compilation times and the sizes of the compiled form, whatever the downstream compiler. Especially, within the ressources allocated (1h CPU time and 7.6 GiB memory), `C2D` has been able to solve 423 instances without any preprocessing technique and 451 instances when `B` + `E` was used first as a preprocessor, while `d4` has been able to solve 422 instances without preprocessing technique and 443 instances when `B` + `E` was used first as a preprocessor.

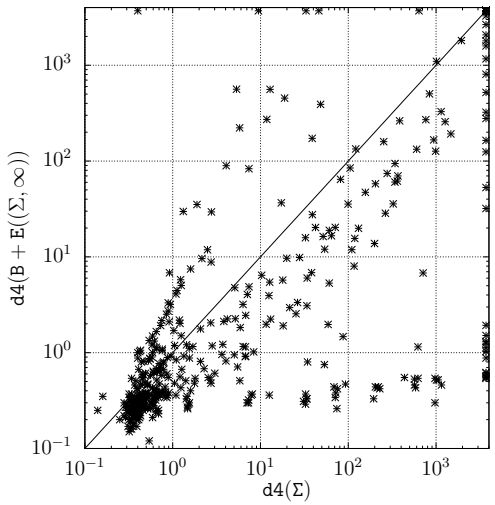
5.4. Projected Model Counting

To close the section, it is worth mentioning that approximate compilation with controllable variables, as described before, is strongly connected to the *projected model counting task*, as considered in [38, 39, 61].

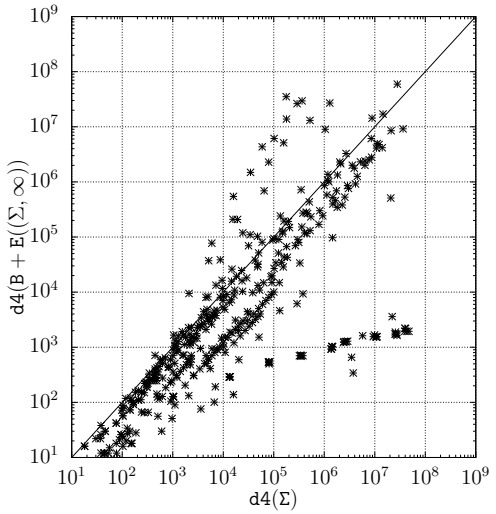


(a) time (in seconds)

(b) size (in number of edges)



(c) time (in seconds)



(d) size (in number of edges)

Figure 8: Evaluation of the preprocessor B + E on planning benchmarks where a set of variables is protected.

Projected model counting asks to count solutions of a Boolean representation Σ with respect to a given set of projected variables C, where multiple solutions that are identical when restricted to the projected variables count as only one solution. Stated otherwise, the objective is to compute $\|\exists(\mathit{Var}(\Sigma) \setminus \mathrm{C}).\Sigma\|$ from Σ and $\mathrm{C} \subseteq$

$\mathcal{P}$. Obviously, the projected model counting problem is a generalization of the model counting one (take C = $Var(\Sigma)$), and its complexity is known to be even harder than the one of #SAT, namely complete for the class #NP [62]. Projected model counting is useful for a number of problems, especially for planning, and also for SAT-based analysis and quantification of information flow in programs [63]. Several projected model counters have been developed accordingly [39].

Clearly enough, generating an approximate compilation of Σ when C is the set of controllable variables is an approach for determining $\|\exists(Var(\Sigma) \setminus C).\Sigma\|$. Indeed, $\|\exists(Var(\Sigma) \setminus C).\Sigma\|$ can be computed in polynomial time from a Decision-DNNF representation of $\exists(Var(\Sigma)\setminus C).\Sigma$ or (equivalently) of $\exists(Var(\Sigma)\setminus C).\mathtt{B} + \mathtt{E}(\Sigma)$, provided that $C \subseteq I$ is ensured. Whenever the equivalence

$$\exists(Var(\Sigma) \setminus C).\Sigma \equiv \exists(Var(\Sigma) \setminus C).\mathtt{B} + \mathtt{E}(\Sigma)$$

holds, $\|\exists(Var(\Sigma) \setminus C).\Sigma\|$ can be easily derived from $\|\exists(Var(\Sigma) \setminus C).\mathtt{B} + \mathtt{E}(\Sigma)\|$ (to get the former, just multiply the latter by 2^r where

$$r = \#((Var(\Sigma) \setminus Var(\mathtt{B} + \mathtt{E}(\Sigma)) \cap C)$$

is the number of variables of C that have been removed from Σ by running the preprocessor $\mathtt{B} + \mathtt{E}$ on it). As a consequence, if C is the set of variables onto which Σ must be projected, then $\mathtt{B} + \mathtt{E}(\Sigma)$ can be used instead of Σ as the input of the projected model counter under consideration, provided that it is ensured that C is a subset of the variables I computed by B. Since the variables from the corresponding set O have anyway to be forgotten from Σ, an alternative to letting the whole forgetting job to be achieved by the projected model counter is to use E in addition during a preprocessing step. Due to the reduction of the instance Σ achieved by the $\mathtt{B} + \mathtt{E}$ preprocessor both with respect to the number of variables and to the number of clauses, it can easily be the case that the performance of the projected model counter used downstream is improved.

More generally, it turns out that, when the goal is "only" to compute the value $\|\exists(Var(\Sigma) \setminus C).\Sigma\|$ but no approximate compilation of Σ on C is expected to be generated, the condition $C \subseteq I$ imposed on the bipartition found by B can be relaxed. Indeed, as a direct consequence of Craig/Lyndon interpolation theorem [64], for every variable $y \in C$, a formula Φ_C is a gate of y on C in $\exists(Var(\Sigma) \setminus C).\Sigma$ if and only if Φ_C is a gate of y on C in Σ. Stated otherwise, the gates of $\exists(Var(\Sigma) \setminus C).\Sigma$ that contain only variables of C are precisely the gates of Σ that contain only variables of C. Thanks to this result, such gates can be identified from Σ without needing to compute explicitly $\exists(Var(\Sigma) \setminus C).\Sigma$, which

would be prohibitive in the general case (variable elimination is often expensive when many variables are to be eliminated). When the gates have been identified, the corresponding output variables y can be forgotten from Σ since we have $\|\exists(Var(\Sigma)\backslash C).\Sigma\| = \|\exists y.(\exists(Var(\Sigma)\backslash C).\Sigma)\| = \|\exists(Var(\Sigma)\backslash C).(\exists y.\Sigma)\|$. The elimination of y in Σ can be achieved by using E or just by removing y from C and letting the projected model counter finish the job. Interestingly, B can be easily modified to identify gates on C with output variables in C: at line 2, it is enough to replace $Var(\Sigma)$ by $Var(\Sigma)\cap C$ (which is the set of variables of $\exists(Var(\Sigma)\setminus C).\Sigma$). Once a definability bipartition of the variables of $Var(\Sigma)\cap C$ has been computed, there are two options: either B stops and returns the set of output variables, or the loop at line 4 is executed once more, V being set this time to $Var(\Sigma)\setminus C$; in the latter case, gates on $Var(\Sigma)$ with output variables in $Var(\Sigma)\setminus C$ can be identified, so that the corresponding output variables can be (tentatively) eliminated using E. Finally, the output variables $y \in Var(\Sigma)\setminus C$ that have been found by B and not eliminated by E can be exploited when the projected model counter considered downstream is based on top-down search with detection of disjoint components: though, in the general case, one has $\|\exists y.\Sigma\| \neq \|\Sigma \mid \neg y\| + \|\Sigma \mid y\|$, the equation $\|\exists y.\Sigma\| = \|\Sigma \mid \neg y\| + \|\Sigma \mid y\|$ holds whenever y is defined in terms of $Var(\Sigma)\setminus\{y\}$ in Σ. Thus, the constraint of searching on priority variables first (those of C) that is at the core of some projected model counters (especially, Dsharp$_P$) [39] can be relaxed by adding to the priority variables those that are definable in the current formula. This gives to the projected model counter the opportunity of being more efficient via an earlier discovery of disjoint components.

6. Other Related Work

Before concluding, let us compare our approach with some additional related work, concerning the two key mechanisms at work in B + E, namely variable elimination and definability bipartition.

6.1. Variable Elimination

Variable elimination (i.e., computing $\exists X.\Sigma$ from X and Σ) is a fundamental automated reasoning task. First of all, it is the approach on which the good old Davis-Putnam's algorithm for solving SAT was based [43]. In AI, it has many applications in various domains (including planning, diagnosis, belief update, reasoning under inconsistency, etc., see [42] for details). In the area of formal verification, variable elimination also finds applications in unbounded model checking,

information flow analysis, predicate abstraction, etc (see e.g., [65]). In car configuration, it proves useful for projecting the feasible solutions (the models of a given constraint) onto the customer codes (i.e., forgetting the manufacturer control codes) [66]. More generally, in every domain where propositional encodings are exploited, variable elimination is an important issue. Thus, it is not surprising that variable elimination gave rise to an extensive literature and that a number of approaches have been proposed so far for this purpose.

Some approaches take advantage of the fact that variable elimination can be achieved in polynomial time when the input formula Σ is a DNF formula. Basically, starting with a CNF formula Σ, those approaches consist in enumerating (using a SAT solver) implicants of Σ which are then projected onto $Var(\Sigma) \setminus X$, resulting in terms γ over $Var(\Sigma) \setminus X$ (see [67, 68]). By construction, each such γ is an implicant of $\exists X.\Sigma$. Each time such a γ is found, a clause equivalent to $\neg\gamma$ is added to Σ and the process is repeated up to exhaustion (i.e., when the resulting Σ becomes inconsistent). The disjunction of all the γ that have been generated is a DNF formula D equivalent to $\exists X.\Sigma$. In order to keep the size of D sufficiently small (if possible), focusing on prime implicants (as in [69]) or even on shortest implicants (as in [65]) prove useful.

If a CNF formula C equivalent to $\exists X.\Sigma$ is expected, then the DNF formula D must be turned into CNF. Several techniques can be used to this end. One of them consists in computing incrementally an OBDD representation of the DNF formula D: starting with an OBDD representation R equivalent to $\top$ (R consists of a single node 1), each time a term γ is generated, an OBDD representation of it is disjoined with the current OBDD representation R, leading to a new OBDD representation [70]. The time needed by each step is in $\mathcal{O}(|\gamma| \cdot |R|)$. Then, when no additional γ is to be considered, the OBDD representation R is turned into a CNF formula, by following all paths leading to the 0 sink (corresponding to terms which are implicants of $\neg\exists X.\Sigma$) and generating their negations (which are thus implicates of $\exists X.\Sigma$). Their conjunction is, by construction, a CNF formula C equivalent to $\exists X.\Sigma$.

Another approach to compute C from D consists in exploiting duality. First, since D is a DNF formula, we can compute in linear time from it a CNF representation of $\neg$D. Then we can apply once more the same process as above, with $X = \emptyset$ and $\neg$D instead of Σ. Thus, one generates using a SAT solver a DNF representation of $\neg$D, by enumerating some implicants of it. Finally, we compute in linear time from it a CNF representation of $\neg\neg$D, that is a CNF representation C of D.

Because the languages DNF and CNF are incomparable with respect to succinctness [71], it can be the case that the translation to CNF leads to a formula

the size of which is exponentially larger (resp. smaller) than the size of the DNF formula. Furthermore, when one takes advantage of the OBDD-based approach, since the language OBDD is incomparable with respect to succinctness with DNF, it can also be the case that the OBDD obtained as an intermediate representation has a size which is exponentially larger (resp. smaller) than the size of the DNF formula, and since OBDD is also incomparable with respect to succinctness with CNF, it can also be the case that the size of the output CNF is exponentially larger (resp. smaller) than the size of the OBDD representation.

While all those approaches can be used for the variable elimination step, it turns out that it would not make sense to take advantage of any of them given our very objective (counting the number of models of $\exists X.\Sigma$). Indeed, those approaches are not incremental in essence (the variables are eliminated as a whole, and not one by one). Clearly enough, it would not make sense to eliminate the variables incrementally using them (by repeating the elimination process focusing on a single variable at each step), since as soon as a DNF formula equivalent to $\exists x.\Sigma$ has been generated (with x the first variable of X to be eliminated), the elimination of all the other variables of X can be achieved easily. Incrementality is important since one wants to possibly avoid eliminating a variable would it be too computationally expensive to do so. More importantly, the computational effort spent for computing D is much more than one expects for a preprocessing technique for model counting. Indeed, the DNF formula D is by construction an irredundant, yet deterministic implicant cover of $\exists X.\Sigma$: on the one hand, removing a term from D might not preserve the equivalence $D \equiv \exists X.\Sigma$, and on the other hand the terms in D are pairwise inconsistent (if $gamma_i$ denotes the term generated at step i, then each γ_i is an implicant of $\neg\gamma_1 \wedge \ldots \wedge \neg\gamma_{i-1}$). As a consequence, $\|D\| = \|\exists X.\Sigma\|$ can be computed in polynomial time from D. Accordingly, those approaches can be considered as compilation-based techniques for model counting.

Other approaches rely on a substitute-then-simplify principle. Basically, applying this principle consists in considering the variables x of X successively and for each of them, to replace Σ by $(\Sigma \mid \neg x) \vee (\Sigma \mid x)$ once simplified (using the laws of Boolean calculus). By construction, the formula obtained once all the variables have been eliminated is equivalent to $\exists X.\Sigma$. Clearly enough, such approaches are incremental ones but in spite of it, they are not suited to our purpose: they do not ensure that the output formula $\exists X.\Sigma$ is in CNF format or in DNF format (which is a requirement here since one wants to use a model counter downstream, and existing model counters take CNF formulae as inputs). Indeed, when Σ is a CNF formula, the formula generated is a disjunction of CNF formulae. Turning

it into an equivalent CNF formula (or an equivalent DNF formula) may lead to an exponential blowup (i.e., in the worst case, the size of the resulting formula is exponential in the size of the given disjunction of CNF formulae).

A fundamental difference of our resolution-based algorithm E for variable elimination compared to all those approaches is that E is not asked to always succeed in eliminating all the given variables. Indeed, a variable x from X = O is eliminated by E only if its elimination does not lead to increase the number of clauses of the input CNF formula Σ. In order to maintain this number small enough, some efficient preprocessing techniques (vivification and focused occurrence simplification) as well as a postprocessing technique (the removal of subsumed resolvents) are exploited. Since variable elimination is incremental in our approach, one can take advantage of an elimination ordering; the chosen ordering promotes the selection of variables minimizing the number of resolvents that could be generated. The selection heuristics used is actually dynamic in the sense that the elimination of a variable is possibly postponed.

6.2. Definability Bipartition

Finally, a paper describing an approach closely related to our own one, is [40]. In this work, the authors present a smart preprocessing technique for computing an *approximation* of the number of models of a propositional formula Σ, with some approximation guarantees. This goes through a notion of *independent support* of Σ. While the authors do not report any explicit connection with the notion of definability, it turns out that a subset I of variables of $Var(\Sigma)$ is an independent support of Σ if and only if $\langle \mathrm{I}, Var(\Sigma) \setminus \mathrm{I} \rangle$ is a definability bipartition of Σ. Thus, the authors are interested in computing a definability bipartition of Σ (just like us in this paper) but not for the same purpose (one wants to compute the exact number of models of Σ). The key of their approach is that affine clauses over I (i.e., variables of I connected using occurrences of the XOR connective) are 3-universal hash functions. As such, they can be exploited to compute an (ϵ, δ)-approximation of $\|\Sigma\|$, and this can be achieved without eliminating any variable of $Var(\Sigma) \setminus \mathrm{I}$ in Σ (there is no need of E or any similar algorithm for this task). Basically, the approach consists in focusing on randomly chosen parts[11] of the search space (the set of all interpretations over $Var(\Sigma)$) characterized by the interpretations that satisfy the conjunction of the affine clauses that are considered; multiplying the number of parts by the median number of the number of models of Σ in each of

[11] Of course, sufficiently many parts must be considered to ensure the expected confidence $1-\delta$.

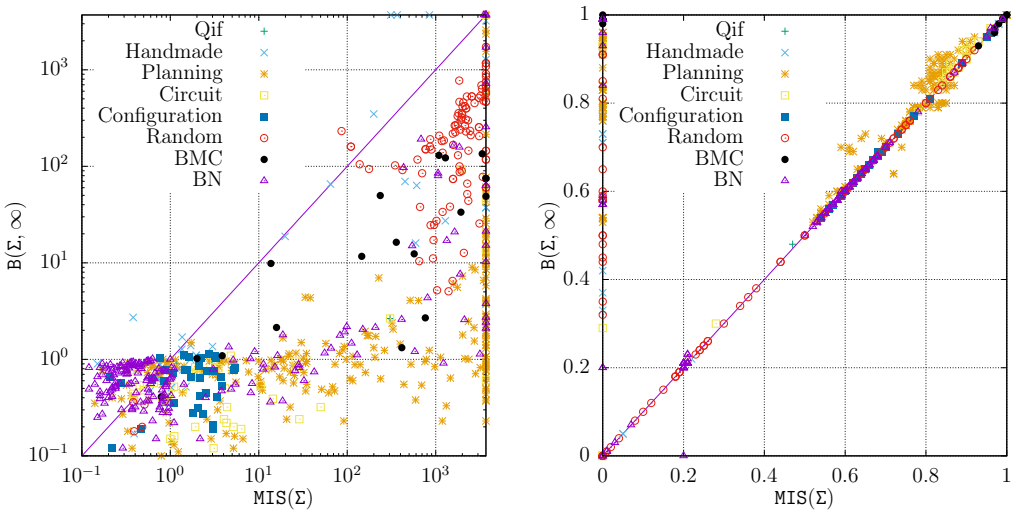
those parts then gives an estimate of $\|\Sigma\|$. In practice, considering affine clauses with low density (i.e., based on a "small" set I) leads to much improved performances of the solvers used to count the number of models of Σ in each cell. This justifies to look for "small" independent sets.

To do the job, the authors exhibit a reduction from the problem of computing an independent support of a CNF formula Σ to the problem of computing a group-oriented unsatisfiable subset of clauses of a CNF formula. More generally, they show that the problem of computing a subset-minimal (resp. a smallest) independent support of Σ amounts to computing a GMUS, i.e., a group-oriented minimal unsatisfiable subset (resp. an SGMUS, i.e., a minimum-sized group-oriented minimal unsatisfiable subset) of a formula that can be computed in time polynomial in the size of Σ (see [40] for details). Formally, a GMUS (resp. SGMUS) of a set Δ of propositional formulae can be defined as follows (see e.g., [72, 73]):

Definition 5 (GMUS, SGMUS). *Let Δ be the union $(\bigcup_{i=1}^{n} \Delta_i) \cup \Omega$ of (finite) sets of propositional formulae, forming a partition, such that Δ is unsatisfiable.*

- *A* group-oriented minimal unsatisfiable subset (GMUS) *(also referred to as high-level MUS) of Δ is a set of formulae $\Phi = (\bigcup_{i=1}^{n} \Phi_i) \cup \Omega$ such that:*
  - $\forall i \in \{1, \ldots, n\}, \Phi_i \subseteq \Delta_i$,
  - Φ *is unsatisfiable,*
  - $\forall i \in \{1, \ldots, n\}, \Phi \setminus \Phi_i$ *is satisfiable.*
- *A* minimum-sized group-oriented minimal unsatisfiable subset (SGMUS) *of Δ is a GMUS of Δ of minimal cardinality.*

The reduction used in [40] can be viewed as an extension of Padoa's approach (as reflected by Theorem 2) for checking the definability in Σ on I of the whole set of variables $Var(\Sigma) \setminus I$ (instead of doing it one variable after the other, as it is the case in B). Since the concept of MUS (and its "neighborhood") has given rise to an abundant literature (together with the implementation of several solvers) for the past few years (see e.g., [72, 74, 75, 73, 52, 76, 77, 78]), the reduction gives the opportunity to take advantage of GMUS and SGMUS extractors for implementing pieces of software for computing independent supports of Σ that are subset-minimal independent supports or among the smallest independent supports. Thus, the solver MIS for computing subset-minimal independent supports is based on



(a) time for finding a definability bipartition $\langle \mathrm{I}, \mathrm{O} \rangle$ of Σ

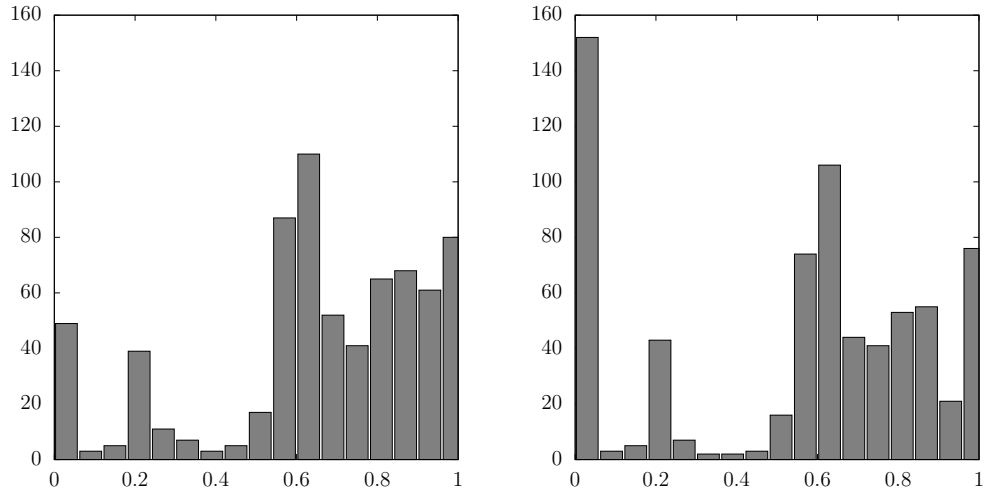
(b) ratio $\frac{|\mathrm{O}|}{|Var(\Sigma)|}$

Figure 9: Comparison of the performance of the bipartitioners `MIS` and `B`.

the GMUS extractor `MUSer2` [73], while the solver `SMIS` for computing smallest independent supports is based on the SGMUS extractor `forqes` [78].

Using such a reduction has some pros and some cons. On the one hand, the implementation effort is reduced. On the other hand, there is no way to set (even partially) the ordering with respect to which the variables of Σ will be tested for definability, i.e., as members of O. In order to compare the performances of `B` and `MIS`, we have performed some additional experiments based on the 703 `CNF` instances described before. The obtained results are reported in the two scatter plots given in Figure 9. As usual, each point in those plots corresponds to a given `CNF` instance Σ. On Figure 9a where the scales of both axes are logarithmic, the x-coordinate of a point Σ is the time (in seconds) needed by `B` to compute a definability bipartition of Σ, while the y-coordinate of Σ is the time needed by `MIS` to do the same job. Figure 9b is about the proportion of variables found as output variables by both approaches: the x-coordinate (resp. y-coordinate) of a point Σ is the value of the ratio $\frac{|\mathrm{O}|}{|Var(\Sigma)|}$ when O is the set of output variables found by `B` (resp. by `MIS`).

Figure 9a shows, on the one hand, that `B` is significantly faster than `MIS`. Figure 9b shows, on the other hand, that the number of output variables (once normalized) found by the two bipartitioners are quite close whenever the com-



(a) $\texttt{B}(\Sigma, \infty)$ (b) $\texttt{MIS}(\Sigma)$

Figure 10: Ratio $\frac{|\mathrm{O}|}{|Var(\Sigma)|}$ for B and MIS.

putation terminated. However, it finished in due time much more often when B was used than when MIS was used. Interestingly, Figure 9b also shows that the proportion of output variables exceeds 50% of the total number of variables for many benchmarks, which explains why B + E is a useful preprocessing technique in practice.

Finally, Figures 10a and 10b reports histograms that indicate (respectively) for each of the two partitioners the number of instances Σ (the heights of the rectangles on the y-axis) for which a ratio of $\frac{|\mathrm{O}|}{|Var(\Sigma)|}$ variables (given on the x-axis) has been found.

Two main observations can be made from Figure 10. On the one hand, the number of instances that have not been solved by MIS in due time is much larger than the corresponding number for B. This coheres with what was observed on Figure 9. On the other hand, B proves to be slightly better than MIS in the sense that it often found slightly more output variables than MIS, especially when this number is huge. Finally, Figure 10 shows more clearly than Figure 9b that a large ratio of output variables (let us say, $> 50\%$) can be found in a large number of benchmarks (MIS found 466 such instances (over 703), while B found 577 such instances).

7. Conclusion

We have defined and evaluated a new preprocessing technique for counting the models of a propositional CNF formula Σ. This technique consists in determining first a definability bipartition of Σ into a set I of input variables and a set O of output variables, then to eliminate in Σ some variables of O. The soundness of this technique is based on standard theorems in classical logic by Beth and Padoa. While those results are quite old, they prove useful for defining a quite effective preprocessing technique to model counting.

More in detail, in order to evaluate the performances of our approach on CNF formulae, we have designed and implemented a preprocessor B + E that is based on it. B + E associates with a given CNF formula Σ a CNF formula B + E(Σ) which has the same number of models as Σ, but is often simpler with respect to the number of variables and size. We tested B + E over many benchmarks from different data sets. Experiments have shown that for many instances Σ, the overall computation time needed to calculate $\|$B + E(Σ)$\|$ using state-of-the art exact model counters is often much lower than the time needed to compute $\|\Sigma\|$ with the same counters. This performance shift can be explained by the two following facts: on the one hand, for many instances, a large number of gates exist; on the other hand, such gates can be detected and eliminated using B + E.

In our experiments, we have also considered the possibility to combine the two preprocessors pmc and B + E in sequence before model counting, in order to determine whether some synergetic effects can be obtained via an improved reduction of the instances. Generally speaking, the results show only slight improvements for the two combinations (pmc then B + E, and B + E then pmc), both in terms of the reductions of the number of variables and of the size of the instances, and also in terms of reduction of the subsequent model counting time (whatever the model counter among the four used). For this reason, and since the paper is already quite long, we refrained from reporting the corresponding results (they can be obtained from the authors on demand).

This work opens a number of perspectives for further research. Considering other heuristics in B for determining a definability bipartition and determining how to tune the constants max#C and max#Res depending on the instance at hand will deserve to be investigated.

Another perspective consists in trying to improve E by exploiting in it some additional preprocessing techniques p. Indeed, it is not mandatory for p to be equivalence-preserving. Actually, p can be exploited as soon as it satisfies the equivalence $\exists$E.p(Σ) $\equiv$ $\exists$E.Σ, which is less demanding than p(Σ) $\equiv$ Σ. This

is the case for instance for some specializations to variables from E of existing preprocessing techniques, such as the pure output literal rule (removing every clause containing a pure literal from E) or the output blocked clause elimination rule (removing every clause containing a variable from E such that every resolvent in Σ obtained by resolving on this variable is a valid clause) [79].

It would be also useful to evaluate the benefits that could be obtained in practice by considering more computationally demanding versions of `B` and of `E` when the objective is to compute an approximate compilation of Σ on a set of controllable variables. Indeed, in such a case, both the `B` step and the `E` step are performed off-line. Thus, it makes sense to spend some extra time (even if it is significant) to compute a "better" definability bipartition (especially, a smallest definability bipartition) and to eliminate more output variables if this leads to a smaller compiled form. Some experiments will be conducted to try to figure out what a good time/space trade-off could be.

Other perspectives concern the interface between definability, model counting and projected model counting. When the objective is to compute $\|\Sigma\|$, instead of taking advantage of `B` + `E` followed by any model counter as discussed before, we could instead use `B` followed by any projected model counter (where the projection is onto I). The other way around, when the objective is to compute $\|\exists(Var(\Sigma) \setminus C).\Sigma\|$, we could refrain from using `B` and exploit `E` directly on $Var(\Sigma) \setminus C$ and Σ as a preprocessing for projected model counting. It would be interesting to implement both approaches and to determine whether they are helpful in practice.

References

[1] T. Sang, P. Beame, H. A. Kautz, Performing Bayesian Inference by Weighted Model Counting, in: M. M. Veloso, S. Kambhampati (Eds.), Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA, AAAI Press / The MIT Press, 2005, pp. 475–482.

[2] M. Chavira, A. Darwiche, On probabilistic inference by weighted model counting, Artif. Intell. 172 (6-7) (2008) 772–799. doi:10.1016/j.artint.2007.11.002.

[3] U. Apsel, R. I. Brafman, Lifted MEU by weighted model counting, in: J. Hoffmann, B. Selman (Eds.), Proceedings of the Twenty-Sixth AAAI

Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada., AAAI Press, 2012, pp. 1861–1867.

[4] A. Choi, D. Kisa, A. Darwiche, Compiling probabilistic graphical models using sentential decision diagrams, in: L. C. van der Gaag (Ed.), Symbolic and Quantitative Approaches to Reasoning with Uncertainty - 12th European Conference, ECSQARU 2013, Utrecht, The Netherlands, July 8-10, 2013. Proceedings, Vol. 7958 of Lecture Notes in Computer Science, Springer, 2013, pp. 121–132. doi:10.1007/978-3-642-39091-3_11.

[5] H. Palacios, B. Bonet, A. Darwiche, H. Geffner, Pruning Conformant Plans by Counting Models on Compiled d-DNNF Representations, in: S. Biundo, K. L. Myers, K. Rajan (Eds.), Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA, AAAI, 2005, pp. 141–150.

[6] C. Domshlak, J. Hoffmann, Fast probabilistic planning through weighted model counting, in: D. Long, S. F. Smith, D. Borrajo, L. McCluskey (Eds.), Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK, June 6-10, 2006, AAAI, 2006, pp. 243–252.

[7] L. Feiten, M. Sauer, T. Schubert, A. Czutro, E. Böhl, I. Polian, B. Becker, #SAT-based vulnerability analysis of security components - A case study, in: 2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT 2012, Austin, TX, USA, October 3-5, 2012, IEEE Computer Society, 2012, pp. 49–54. doi:10.1109/DFT.2012.6378198.

[8] S. Chakraborty, D. Fried, K. S. Meel, M. Y. Vardi, From weighted to unweighted model counting, in: Q. Yang, M. Wooldridge (Eds.), Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, AAAI Press, 2015, pp. 689–695.

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

[10] D. Roth, On the hardness of approximate reasoning, Artif. Intell. 82 (1-2) (1996) 273–302. doi:10.1016/0004-3702(94)00092-1.

[11] M. E. Dyer, L. A. Goldberg, M. Jerrum, The complexity of weighted Boolean #CSP, SIAM J. Comput. 38 (5) (2009) 1970–1986. doi:10.1137/070690201.

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

[13] M. Samer, S. Szeider, Algorithms for propositional model counting, J. Discrete Algorithms 8 (1) (2010) 50–64. doi:10.1016/j.jda.2009.06.002.

[14] F. Bacchus, S. Dalmao, T. Pitassi, Algorithms and Complexity Results for #SAT and Bayesian Inference, in: 44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings, IEEE Computer Society, 2003, pp. 340–351. doi:10.1109/SFCS.2003.1238208.

[15] C. P. Gomes, A. Sabharwal, B. Selman, Model counting, in: A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, Vol. 185 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2009, pp. 633–654. doi:10.3233/978-1-58603-929-5-633.

[16] D. Achlioptas, Z. Hammoudeh, P. Theodoropoulos, Fast and flexible probabilistic model counting, in: Beyersdorff and Wintersteiger [80], pp. 148–164. doi:10.1007/978-3-319-94144-8_10.

[17] D. Achlioptas, P. Theodoropoulos, Probabilistic model counting with short XORs, in: S. Gaspers, T. Walsh (Eds.), Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings, Vol. 10491 of Lecture Notes in Computer Science, Springer, 2017, pp. 3–19. doi:10.1007/978-3-319-66263-3_1.

[18] S. Chakraborty, K. S. Meel, M. Y. Vardi, Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls, in: Kambhampati [81], pp. 3569–3576.

[19] F. Bacchus, J. Winter, Effective preprocessing with hyper-resolution and equality reduction, in: E. Giunchiglia, A. Tacchella (Eds.), Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003.

Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers, Vol. 2919 of Lecture Notes in Computer Science, Springer, 2003, pp. 341–355. doi:10.1007/978-3-540-24605-3_26.

[20] S. Subbarayan, D. K. Pradhan, Niver: Non increasing variable elimination resolution for preprocessing SAT instances, in: SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings [82], pp. 276–291.

[21] I. Lynce, J. P. Marques Silva, Probing-based preprocessing techniques for propositional satisfiability, in: 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2003), 3-5 November 2003, Sacramento, California, USA, IEEE Computer Society, 2003, p. 105. doi:10.1109/TAI.2003.1250177.

[22] N. Eén, A. Biere, Effective preprocessing in SAT through variable and clause elimination, in: F. Bacchus, T. Walsh (Eds.), Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings, Vol. 3569 of Lecture Notes in Computer Science, Springer, 2005, pp. 61–75. doi:10.1007/11499107_5.

[23] C. Piette, Y. Hamadi, L. Sais, Vivifying propositional clausal formulae, in: M. Ghallab, C. D. Spyropoulos, N. Fakotakis, N. M. Avouris (Eds.), ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings, Vol. 178 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2008, pp. 525–529. doi:10.3233/978-1-58603-891-5-525.

[24] H. Han, F. Somenzi, Alembic: An efficient algorithm for CNF preprocessing, in: Proceedings of the 44th Design Automation Conference, DAC 2007, San Diego, CA, USA, June 4-8, 2007, IEEE, 2007, pp. 582–587. doi:10.1145/1278480.1278628.

[25] M. Heule, M. Järvisalo, A. Biere, Clause elimination procedures for CNF formulas, in: C. G. Fermüller, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17, Yogyakarta, Indonesia, October 10-15, 2010. Proceedings, Vol. 6397 of Lecture Notes in Computer Science, Springer, 2010, pp. 357–371. doi:10.1007/978-3-642-16242-8_26.

[26] M. Järvisalo, A. Biere, M. Heule, Simulating circuit-level simplifications on CNF, J. Autom. Reasoning 49 (4) (2012) 583–619. doi:10.1007/s10817-011-9239-9.

[27] M. Heule, M. Järvisalo, A. Biere, Efficient CNF simplification based on binary implication graphs, in: K. A. Sakallah, L. Simon (Eds.), Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings, Vol. 6695 of Lecture Notes in Computer Science, Springer, 2011, pp. 201–215. doi:10.1007/978-3-642-21581-0_17.

[28] G. Audemard, L. Simon, Predicting learnt clauses quality in modern SAT solvers, in: C. Boutilier (Ed.), IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009, 2009, pp. 399–404.

[29] A. Biere, Lingeling essentials, A tutorial on design and implementation aspects of the the SAT solver lingeling, in: D. Le Berre (Ed.), POS-14. Fifth Pragmatics of SAT workshop, a workshop of the SAT 2014 conference, part of FLoC 2014 during the Vienna Summer of Logic, July 13, 2014, Vienna, Austria, Vol. 27 of EPiC Series in Computing, EasyChair, 2014, p. 88.

[30] N. Manthey, Solver description of RISS 2.0 and PRISS 2.0, Tech. rep., TU Dresden, Knowledge Representation and Reasoning (2012).

[31] N. Manthey, Coprocessor 2.0 - A flexible CNF simplifier - (tool presentation), in: Cimatti and Sebastiani [83], pp. 436–441. doi:10.1007/978-3-642-31612-8_34.

[32] J.-M. Lagniez, P. Marquis, On preprocessing techniques and their impact on propositional model counting, J. Autom. Reasoning 58 (4) (2017) 413–481. doi:10.1007/s10817-016-9370-8.
URL https://doi.org/10.1007/s10817-016-9370-8

[33] J. Lang, P. Marquis, On propositional definability, Artif. Intell. 172 (8-9) (2008) 991–1017. doi:10.1016/j.artint.2007.12.003.

[34] G. Tseitin, On the complexity of derivation in propositional calculus, Steklov Mathematical Institute, 1968, Ch. Structures in Constructive Mathematics and Mathematical Logic, pp. 115–125.

[35] D. A. Plaisted, S. Greenbaum, A structure-preserving clause form translation, J. Symb. Comput. 2 (3) (1986) 293–304. doi:10.1016/S0747-7171(86)80028-1.

[36] J.-M. Lagniez, E. Lonca, P. Marquis, Improving model counting by leveraging definability, in: Kambhampati [81], pp. 751–757.

[37] J.-M. Lagniez, P. Marquis, An Improved Decision-DNNF Compiler, in: C. Sierra (Ed.), Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017, ijcai.org, 2017, pp. 667–673. doi:10.24963/ijcai.2017/93.

[38] M. Gebser, B. Kaufmann, T. Schaub, Solution enumeration for projected Boolean search problems, in: W. J. van Hoeve, J. N. Hooker (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27-31, 2009, Proceedings, Vol. 5547 of Lecture Notes in Computer Science, Springer, 2009, pp. 71–86. doi:10.1007/978-3-642-01929-6_7.

[39] R. A. Aziz, G. Chu, C. J. Muise, P. J. Stuckey, #∃SAT: Projected model counting, in: M. Heule, S. Weaver (Eds.), Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings, Vol. 9340 of Lecture Notes in Computer Science, Springer, 2015, pp. 121–137. doi:10.1007/978-3-319-24318-4_10.

[40] A. Ivrii, S. Malik, K. S. Meel, M. Y. Vardi, On computing minimal independent support and its applications to sampling and counting, Constraints 21 (1) (2016) 41–58. doi:10.1007/s10601-015-9204-z.

[41] H. B. Enderton, A mathematical introduction to logic, Academic Press, 1972.

[42] J. Lang, P. Liberatore, P. Marquis, Propositional independence: Formula-variable independence and forgetting, J. Artif. Intell. Res. 18 (2003) 391–443. doi:10.1613/jair.1113.

[43] M. Davis, H. Putnam, A computing procedure for quantification theory, J. ACM 7 (3) (1960) 201–215. doi:10.1145/321033.321034.

[44] E. Beth, On Padoa's method in the theory of definition, Indagationes mathematicae 15 (1953) 330–339.

[45] A. Padoa, Essai d'une théorie algébrique des nombres entiers, précédé d'une introduction logique à une théorie déductive quelconque, in: Bibliothèque du Congrès International de Philosophie, Paris, 1903, pp. 309–365.

[46] S. A. Cook, The complexity of theorem-proving procedures, in: M. A. Harrison, R. B. Banerji, J. D. Ullman (Eds.), Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA, ACM, 1971, pp. 151–158. doi:10.1145/800157.805047.

[47] J. P. Marques Silva, K. A. Sakallah, GRASP - a new search algorithm for satisfiability, in: ICCAD, 1996, pp. 220–227. doi:10.1109/ICCAD.1996.569607.
URL https://doi.org/10.1109/ICCAD.1996.569607

[48] J. P. Marques Silva, K. A. Sakallah, GRASP: A search algorithm for propositional satisfiability, IEEE Trans. Computers 48 (5) (1999) 506–521. doi:10.1109/12.769433.
URL https://doi.org/10.1109/12.769433

[49] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an efficient SAT solver, in: Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001, ACM, 2001, pp. 530–535. doi:10.1145/378239.379017.
URL https://doi.org/10.1145/378239.379017

[50] H. Zhang, M. E. Stickel, An efficient algorithm for unit propagation, in: In Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics (AI-MATH'96), Fort Lauderdale (Florida USA, 1996, pp. 166–169.

[51] O. Roussel, Controlling a solver execution with the runsolver tool, JSAT 7 (4) (2011) 139–144.
URL https://satassociation.org/jsat/index.php/jsat/article/view/90

[52] G. Audemard, J.-M. Lagniez, L. Simon, Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction, in: Järvisalo

and Gelder [84], pp. 309–317. doi:10.1007/978-3-642-39071-5_23.

[53] T. Sang, F. Bacchus, P. Beame, H. A. Kautz, T. Pitassi, Combining component caching and clause learning for effective model counting, in: SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings [82], pp. 20–28.

[54] M. Thurley, sharpSAT - counting models with advanced component caching and implicit BCP, in: A. Biere, C. P. Gomes (Eds.), Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings, Vol. 4121 of Lecture Notes in Computer Science, Springer, 2006, pp. 424–429. doi:10.1007/11814948_38.

[55] A. Darwiche, Decomposable negation normal form, J. ACM 48 (4) (2001) 608–647. doi:10.1145/502090.502091.

[56] A. Darwiche, New advances in compiling CNF into decomposable negation normal form, in: R. L. de Mántaras, L. Saitta (Eds.), Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004, IOS Press, 2004, pp. 328–332.

[57] M. Samer, S. Szeider, Algorithms for propositional model counting, J. Discrete Algorithms 8 (1) (2010) 50–64. doi:10.1016/j.jda.2009.06.002.

[58] M. Cadoli, F. M. Donini, M. Schaerf, R. Silvestri, On compact representations of propositional circumscription, Theor. Comput. Sci. 182 (1-2) (1997) 183–202. doi:10.1016/S0304-3975(96)00182-X.

[59] A. del Val, An analysis of approximate knowledge compilation, in: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes, Morgan Kaufmann, 1995, pp. 830–836.

[60] A. Bart, F. Koriche, J.-M. Lagniez, P. Marquis, Symmetry-driven decision diagrams for knowledge compilation, in: T. Schaub, G. Friedrich,

B. O'Sullivan (Eds.), ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014), Vol. 263 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2014, pp. 51–56. doi:10.3233/978-1-61499-419-0-51.

[61] J. K. Fichte, M. Hecher, M. Morak, S. Woltran, Exploiting treewidth for projected model counting and its limits, in: Beyersdorff and Wintersteiger [80], pp. 165–184. doi:10.1007/978-3-319-94144-8_11.

[62] A. Durand, M. Hermann, P. G. Kolaitis, Subtractive reductions and complete problems for counting complexity classes, Theor. Comput. Sci. 340 (3) (2005) 496–513. doi:10.1016/j.tcs.2005.03.012.

[63] V. Klebanov, N. Manthey, C. J. Muise, SAT-based analysis and quantification of information flow in programs, in: K. R. Joshi, M. Siegle, M. Stoelinga, P. R. D'Argenio (Eds.), Quantitative Evaluation of Systems - 10th International Conference, QEST 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings, Vol. 8054 of Lecture Notes in Computer Science, Springer, 2013, pp. 177–192. doi:10.1007/978-3-642-40196-1_16.

[64] W. Craig, Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory, Journal of Symbolic Logic 22 (1957) 269–285.

[65] J. Brauer, A. King, J. Kriener, Existential quantification as incremental SAT, in: G. Gopalakrishnan, S. Qadeer (Eds.), Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings, Vol. 6806 of Lecture Notes in Computer Science, Springer, 2011, pp. 191–207. doi:10.1007/978-3-642-22110-1_17.

[66] C. Zengler, W. Küchlin, Boolean quantifier elimination for automotive configuration - A case study, in: C. Pecheur, M. Dierkes (Eds.), Formal Methods for Industrial Critical Systems - 18th International Workshop, FMICS 2013, Madrid, Spain, September 23-24, 2013. Proceedings, Vol. 8187 of Lecture Notes in Computer Science, Springer, 2013, pp. 48–62. doi:10.1007/978-3-642-41010-9_4.

[67] J. Brauer, A. King, Approximate Quantifier Elimination for Propositional Boolean Formulae, in: M. G. Bobaru, K. Havelund, G. J. Holzmann,

R. Joshi (Eds.), NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings, Vol. 6617 of Lecture Notes in Computer Science, Springer, 2011, pp. 73–88. doi:10.1007/978-3-642-20398-5_7.

[68] D. Monniaux, Quantifier elimination by lazy model enumeration, in: T. Touili, B. Cook, P. B. Jackson (Eds.), Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings, Vol. 6174 of Lecture Notes in Computer Science, Springer, 2010, pp. 585–599. doi:10.1007/978-3-642-14295-6_51.

[69] R. Schrag, Compilation for critically constrained knowledge bases, in: W. J. Clancey, D. S. Weld (Eds.), Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 1., AAAI Press / The MIT Press, 1996, pp. 510–515.

[70] A. Gupta, Z. Yang, P. Ashar, A. Gupta, SAT-based image computation with application in reachability analysis, in: W. A. H. Jr., S. D. Johnson (Eds.), Formal Methods in Computer-Aided Design, Third International Conference, FMCAD 2000, Austin, Texas, USA, November 1-3, 2000, Proceedings, Vol. 1954 of Lecture Notes in Computer Science, Springer, 2000, pp. 354–371. doi:10.1007/3-540-40922-X_22.

[71] A. Darwiche, P. Marquis, A knowledge compilation map, J. Artif. Intell. Res. 17 (2002) 229–264. doi:10.1613/jair.989.

[72] A. Nadel, Boosting minimal unsatisfiable core extraction, in: R. Bloem, N. Sharygina (Eds.), Proceedings of 10th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2010, Lugano, Switzerland, October 20-23, IEEE, 2010, pp. 221–229.
URL http://ieeexplore.ieee.org/document/5770953/

[73] A. Belov, J. P. Marques Silva, Muser2: An efficient MUS extractor, JSAT 8 (3/4) (2012) 123–128.

[74] M. Janota, J. P. Marques Silva, On deciding MUS membership with QBF, in: J. H. Lee (Ed.), Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September

12-16, 2011. Proceedings, Vol. 6876 of Lecture Notes in Computer Science, Springer, 2011, pp. 414–428. doi:10.1007/978-3-642-23786-7_32.

[75] A. Belov, A. Ivrii, A. Matsliah, J. P. Marques Silva, On efficient computation of variable MUSes, in: Cimatti and Sebastiani [83], pp. 298–311. doi:10.1007/978-3-642-31612-8_23.

[76] J.-M. Lagniez, A. Biere, Factoring out assumptions to speed up MUS extraction, in: Järvisalo and Gelder [84], pp. 276–292. doi:10.1007/978-3-642-39071-5_21.

[77] J. P. Marques Silva, A. Previti, On Computing Preferred MUSes and MCSes, in: C. Sinz, U. Egly (Eds.), Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings, Vol. 8561 of Lecture Notes in Computer Science, Springer, 2014, pp. 58–74. doi:10.1007/978-3-319-09284-3_6.

[78] A. Ignatiev, A. Previti, M. H. Liffiton, J. P. Marques Silva, Smallest MUS extraction with minimal hitting set dualization, in: G. Pesant (Ed.), Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings, Vol. 9255 of Lecture Notes in Computer Science, Springer, 2015, pp. 173–182. doi:10.1007/978-3-319-23219-5_13.

[79] O. Kullmann, On a generalization of extended resolution, Discrete Applied Mathematics 96-97 (1999) 149–176. doi:10.1016/S0166-218X(99)00037-2.

[80] O. Beyersdorff, C. M. Wintersteiger (Eds.), Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings, Vol. 10929 of Lecture Notes in Computer Science, Springer, 2018. doi:10.1007/978-3-319-94144-8.

[81] S. Kambhampati (Ed.), Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016, IJCAI/AAAI Press, 2016.

[82] SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings.

[83] A. Cimatti, R. Sebastiani (Eds.), Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings, Vol. 7317 of Lecture Notes in Computer Science, Springer, 2012. doi:10.1007/978-3-642-31612-8.

[84] M. Järvisalo, A. V. Gelder (Eds.), Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings, Vol. 7962 of Lecture Notes in Computer Science, Springer, 2013. doi:10.1007/978-3-642-39071-5.