Improving Model Counting by Leveraging Definability

Jean-Marie Lagniez and **Emmanuel Lonca** and **Pierre Marquis**
CRIL-CNRS and Université d'Artois, Lens, France
email:{lagniez, lonca, marquis}@cril.univ-artois.fr

Abstract

We present 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.

1 Introduction

Propositional model counting (alias the #SAT problem) is the task consisting in computing the number of models of a given propositional formula Σ. This problem and its direct generalization, weighted model counting, are central to many AI problems including probabilistic inference [Sang *et al.*, 2005; Chavira and Darwiche, 2008; Apsel and Brafman, 2012; Choi *et al.*, 2013] and forms of planning [Palacios *et al.*, 2005; Domshlak and Hoffmann, 2006]. 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., [Feiten *et al.*, 2012]). However, propositional model counting is computationally hard (a #P-complete problem), actually much harder in practice than the satisfiability issue (the SAT problem). Its significance 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 [Samer and Szeider, 2010; Bacchus *et al.*, 2003; Gomes *et al.*, 2009].

In this paper, we present a new preprocessing technique for improving exact model counting. Preprocessing techniques are nowadays acknowledged as computationally valuable for a number of automated reasoning tasks, especially SAT solving and QBF solving [Bacchus and Winter, 2004; Subbarayan and Pradhan, 2004; Lynce and Marques-Silva, 2003; Een and Biere, 2005; Piette *et al.*, 2008; Han and Somenzi, 2007; Heule *et al.*, 2010; Järvisalo *et al.*, 2012; Heule *et al.*, 2011]. As such, they are now embodied in some state-of-the-art SAT solvers, like `Glucose` [Audemard and Simon, 2009] which takes advantage of the `SatELite` preprocessor [Een and Biere, 2005], `Lingeling` [Biere, 2014] which has an internal preprocessor, and `Riss` [Manthey, 2012b] which takes advantage of the `Coprocessor` preprocessor [Manthey, 2012a].

Our approach elaborates on [Lagniez and Marquis, 2014], which describes a number of preprocessing techniques that can be exploited for improving the model counting task, computationally speaking. Among them is *gate detection and replacement*. Basically, every variable y of the input formula Σ which turns out to be defined in Σ in terms of other variables $\mathrm{X} = \{\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{k}\}$ can be replaced by its definition 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 every model of Σ which extends this partial assignment gives to y the same truth value. In [Lagniez and Marquis, 2014], literal equivalences, AND/OR gates and XOR gates are detected (either syntactically or using Boolean Constraint Propagation). The empirical results reported in [Lagniez and Marquis, 2014] about the preprocessor `pmc` equipped with the so-called *#eq* combination of preprocessings 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, our preprocessing technique to model counting aims at exploiting in a much more aggressive way the existence of gates within the input formula Σ. The key idea of our approach is that *one does not need to identify the gates themselves but it can be enough to determine that such gates exist*. To be more precise, it proves sufficient to detect that some definability relations between variables hold, without needing to identify the corresponding definitions. This distinction is of tremendous importance since on the one hand, the search space for the possible definitions Φ_X is very large (2^{2^k} elements up to logical equivalence, when X contains k variables), and on the other hand, in the general case, the size of any explicit definition 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) [Lang and Marquis, 2008].

Thus, in the following, we describe a new preprocessor `B + E` which associates with a given `CNF` formula[1] Σ a `CNF` formula Φ which has the same number of models as Σ, but is at least as simple as Σ w.r.t. the number of variables and the size. `B + E` consists of two parts: `B` which aims at determining 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 at *Eliminating* in Σ some variables of O. Our contribution mainly consists of the presentation of the algorithms `B` and `E`, a property establishing the correctness of our preprocessor, and some empirical results showing the computational improvements for model counting offered by `B + E` compared to `pmc`. The benchmarks used, the implementation (runtime code) of `B + E`, and some detailed empirical results are available on line from `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 it is correct. Section 4 presents results from our large scale experiments, showing `B + E` as a challenging preprocessor for model counting, especially when compared with `pmc`. Finally, Section 5 concludes the paper and lists some perspectives for further research.

2 On Definability

Let $\mathcal{L}$ be the (classical) propositional language defined inductively from a countable 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. $\models$ denotes logical entailment and $\equiv$ logical equivalence. For any formula Σ from $\mathcal{L}$, $Var(\Sigma)$ is the set of variables from $\mathcal{P}$ occurring in Σ, and $\|\Sigma\|$ is the number of models of Σ over $Var(\Sigma)$. A literal ℓ is a variable $\ell = x$ from $\mathcal{P}$ or a negated one $\ell = \neg x$. When ℓ is a literal, $var(\ell)$ denotes the variable upon which ℓ is built. A term is a conjunction of literals or $\top$, and a clause is a disjunction of literals or $\bot$. A `CNF` formula is a conjunction of clauses. Let X be any subset of $\mathcal{P}$. A canonical term γ_X over X is a consistent term into which every variable from X appears (either as a positive literal or as a negative one, i.e., as a negated variable). $\exists X.\Sigma$ denotes any formula from $\mathcal{L}$ equivalent to the forgetting of X in Σ, i.e., the strongest logical consequence of Σ which is independent of variables from X.

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

Definition 1 (implicit definability) *Let $\Sigma \in \mathcal{L}$, $X \subseteq \mathcal{P}$ and $y \in \mathcal{P}$. Σ* implicitly defines y *in terms of X if and only if for every canonical term γ_X over X, we have $\gamma_X \wedge \Sigma \models y$ or $\gamma_X \wedge \Sigma \models \neg y$.*

Definition 2 (explicit definability) *Let $\Sigma \in \mathcal{L}$, $X \subseteq \mathcal{P}$ and $y \in \mathcal{P}$. Σ* explicitly defines y *in terms of X if and only if there exists a formula $\Phi_X \in PROP_X$ s.t. $\Sigma \models \Phi_X \leftrightarrow y$. 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.*

Example 1 *Let Σ be the `CNF` formula consisting of the following clauses:*

$a \vee b$,	$\neg a \vee \neg b \vee d$,	$a \vee e$,
$a \vee c \vee \neg e$,	$\neg a \vee \neg c \vee d$,	$b \vee c \vee e$,
$a \vee \neg d$,	$\neg a \vee \neg b \vee c \vee \neg e$,	$\neg b \vee \neg c \vee e$.
$b \vee c \vee \neg d$,	$\neg a \vee b \vee \neg c \vee \neg e$,	

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

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

What happens in this example is not fortuitous due to the following theorem from [Beth, 1953]:

Theorem 1 *Let $\Sigma \in \mathcal{L}$, $X \subseteq \mathcal{P}$ and $y \in \mathcal{P}$. Σ implicitly defines 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. An interesting consequence of this theorem is that it is not mandatory to point out a definition Φ_X of y in terms of X in order to prove that such a definition 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 [Lang and Marquis, 2008]. To prove it, we can take advantage of the following result (Padoa's theorem [Padoa, 1903]), restricted to propositional logic and recalled in [Lang and Marquis, 2008]; this theorem gives an entailment-based characterization of (implicit) definability:

Theorem 2 *For any $\Sigma \in \mathcal{L}$ and any $X \subseteq \mathcal{P}$, let Σ'_X be the formula obtained by replacing in Σ in a uniform way every propositional symbol z from $Var(\Sigma) \setminus X$ by a new propositional symbol 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 Preprocessor to Model Counting

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. To be more precise, the objective is first to find (if possible) a definability bipartition $\langle I, O\rangle$ of Σ where I contains as few elements as possible.

[1]Requiring the input to be a `CNF` formula is not a major restriction since Tseitin transformation [Tseitin, 1968] can be used to turn any propositional circuit into a `CNF` formula which has the same number of models – indeed, the variables which are introduced are actually defined from the original ones and the transformation consists in adding gates to the input. Interestingly, the `CNF` format is the one considered by state-of-the-art model counters.

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

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

Then in a second step, variables from O are forgotten in Σ so as to simplify it. This leads to the preprocessing algorithm `B + E` (*B(ipartition), then E(liminate)*) given at Algorithm 1:

Algorithm 1: `B + E`

---

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

$\mathrm{O} \leftarrow \mathtt{B}(\Sigma)$;
$\Phi \leftarrow \mathtt{E}(\mathrm{O}, \Sigma)$;
return Φ

Interestingly, both steps in this algorithm can be tuned in order to keep the preprocessing phase light from a computational standpoint. On the one hand, it is not necessary to determine a definability bipartition $\langle \mathrm{I}, \mathrm{O} \rangle$ of Σ for which the cardinality of I is minimal (identifying a reasonable amount of output variables can prove sufficient). On the other hand, it is not necessary to forget (i.e., eliminate) in Σ every variable from O but focusing on a subset $\mathrm{E} \subseteq \mathrm{O}$ is enough. Formally, our approach is based on the following result, which establishes the correctness of `B + E`:

Proposition 1 *Let* $\Sigma \in \mathcal{L}$. *Let* $\langle \mathrm{I}, \mathrm{O} \rangle$ *be a definability bipartition of* $\mathit{Var}(\Sigma)$. *Let* $\mathrm{E} \subseteq \mathrm{O}$. *Then* $\|\Sigma\| = \|\exists \mathrm{E}.\Sigma\|$.

Proof: Let $\mathrm{E} = \{\mathrm{y}_1, \ldots, \mathrm{y}_\mathrm{m}\}$ be a subset of O. Since every y_i ($\mathrm{i} \in 1, \ldots, \mathrm{m}$) is definable in terms of I in Σ, there exists a formula $\Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}$ over I such that $\Sigma \models \mathrm{y}_\mathrm{i} \leftrightarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}$ (i.e., a definition of y_i on I in Σ). Let $\Sigma[\mathrm{y}_\mathrm{i} \leftarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}]_{\mathrm{i} \in 1,\ldots,\mathrm{m}}$ be the formula obtained by replacing in Σ every occurrence of y_i by $\Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}$. Let γ_I be any canonical term over I. If $\gamma_\mathrm{I} \wedge \Sigma$ is consistent, then there exists a unique model $\omega_{\gamma_\mathrm{I}}$ of Σ over $\mathit{Var}(\Sigma)$ which 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 $\gamma_\mathrm{I} \wedge \Sigma$ is consistent.

Now, by construction, we have $\Sigma \equiv \bigwedge_{\mathrm{i}=1}^{\mathrm{m}} (\mathrm{y}_\mathrm{i} \leftrightarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}) \wedge \Sigma[\mathrm{y}_\mathrm{i} \leftarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}]_{\mathrm{i} \in 1,\ldots,\mathrm{m}}$. Hence $\gamma_\mathrm{I} \wedge \Sigma$ is equivalent to $\gamma_\mathrm{I} \wedge \bigwedge_{\mathrm{i}=1}^{\mathrm{m}} (\mathrm{y}_\mathrm{i} \leftrightarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}) \wedge \Sigma[\mathrm{y}_\mathrm{i} \leftarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}]_{\mathrm{i} \in 1,\ldots,\mathrm{m}}$. Since γ_I is a canonical term over I and $\mathit{Var}(\Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}) \subseteq \mathrm{I}$ for every $\mathrm{i} \in 1, \ldots, \mathrm{m}$, we have that $\gamma_\mathrm{I} \wedge \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}$ is consistent if and only if $\gamma_\mathrm{I} \models \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}$, so that $\gamma_\mathrm{I} \wedge \bigwedge_{\mathrm{i}=1}^{\mathrm{m}} (\mathrm{y}_\mathrm{i} \leftrightarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}})$ is equivalent to $\gamma_\mathrm{I} \wedge \bigwedge_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{y}_\mathrm{i}^*$) where y_i^* ($\mathrm{i} \in 1, \ldots, \mathrm{m}$) is y_i when $\gamma_\mathrm{I} \models \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}$ and is $\neg \mathrm{y}_\mathrm{i}$ otherwise. Therefore, $\gamma_\mathrm{I} \wedge \bigwedge_{\mathrm{i}=1}^{\mathrm{m}} (\mathrm{y}_\mathrm{i} \leftrightarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}) \wedge \Sigma[\mathrm{y}_\mathrm{i} \leftarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}]_{\mathrm{i} \in 1,\ldots,\mathrm{m}}$ is equivalent to $\gamma_\mathrm{I} \wedge (\bigwedge_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{y}_\mathrm{i}^*) \wedge \Sigma[\mathrm{y}_\mathrm{i} \leftarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}]_{\mathrm{i} \in 1,\ldots,\mathrm{m}}$. In addition, since $\mathit{Var}(\gamma_\mathrm{I} \wedge \Sigma[\mathrm{y}_\mathrm{i} \leftarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}]_{\mathrm{i} \in 1,\ldots,\mathrm{m}}) \subseteq \mathrm{I}$, $\mathit{Var}(\bigwedge_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{y}_\mathrm{i}^*) \subseteq \mathrm{O}$ and $\mathrm{I} \cap \mathrm{O} = \emptyset$, $\gamma_\mathrm{I} \wedge \Sigma$ is consistent if and only if $\gamma_\mathrm{I} \wedge \Sigma[\mathrm{y}_\mathrm{i} \leftarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}]_{\mathrm{i} \in 1,\ldots,\mathrm{m}}$ is consistent. But since γ_I is a canonical term over I and $\mathit{Var}(\Sigma[\mathrm{y}_\mathrm{i} \leftarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}]_{\mathrm{i} \in 1,\ldots,\mathrm{m}}) \subseteq \mathrm{I}$, this is precisely the case when $\gamma_\mathrm{I} \models \Sigma[\mathrm{y}_\mathrm{i} \leftarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}]_{\mathrm{i} \in 1,\ldots,\mathrm{m}}$. Thus we get that the number of canonical terms γ_I over I such that $\gamma_\mathrm{I} \wedge \Sigma$ is consistent is equal to the number of models of $\Sigma[\mathrm{y}_\mathrm{i} \leftarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}]_{\mathrm{i} \in 1,\ldots,\mathrm{m}}$ over I. Stated otherwise, we have $\|\Sigma\| = \|\Sigma[\mathrm{y}_\mathrm{i} \leftarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}]_{\mathrm{i} \in 1,\ldots,\mathrm{m}}\|$.

Since $\mathit{Var}(\Sigma[\mathrm{y}_\mathrm{i} \leftarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}]_{\mathrm{i} \in 1,\ldots,\mathrm{m}}) \cap \mathrm{E} = \emptyset$, we have that $\exists \mathrm{E}.\Sigma \equiv \exists \mathrm{E}.((\bigwedge_{\mathrm{i}=1}^{\mathrm{m}} (\mathrm{y}_\mathrm{i} \leftrightarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}) \wedge \Sigma[\mathrm{y}_\mathrm{i} \leftarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}]_{\mathrm{i} \in 1,\ldots,\mathrm{m}}) \equiv (\exists \mathrm{E}.(\bigwedge_{\mathrm{i}=1}^{\mathrm{m}} (\mathrm{y}_\mathrm{i} \leftrightarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}))) \wedge \Sigma[\mathrm{y}_\mathrm{i} \leftarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}]_{\mathrm{i} \in 1,\ldots,\mathrm{m}}$. Finally, since $\mathit{Var}(\Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}) \cap \mathrm{E} = \emptyset$, we also have that $\exists \mathrm{E}.(\bigwedge_{\mathrm{i}=1}^{\mathrm{m}} (\mathrm{y} \leftrightarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}))$ is equivalent to $\bigwedge_{\mathrm{i}=1}^{\mathrm{m}} (\exists \mathrm{y}_\mathrm{i}.(\mathrm{y}_\mathrm{i} \leftrightarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}))$. But each $\exists \mathrm{y}_\mathrm{i}.(\mathrm{y}_\mathrm{i} \leftrightarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}})$ ($\mathrm{i} \in 1, \ldots, \mathrm{m}$) is a valid formula. Hence we have $\exists \mathrm{E}.\Sigma \equiv \Sigma[\mathrm{y}_\mathrm{i} \leftarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}]_{\mathrm{i} \in 1,\ldots,\mathrm{m}}$, which implies that $\|\exists \mathrm{E}.\Sigma\| = \|\Sigma[\mathrm{y}_\mathrm{i} \leftarrow \Phi_\mathrm{I}^{\mathrm{y}_\mathrm{i}}]_{\mathrm{i} \in 1,\ldots,\mathrm{m}}\|$, and thus that $\|\Sigma\| = \|\exists \mathrm{E}.\Sigma\|$. ■

The ability to identify only a subset O of output variables in the bipartition generation phase, and to consider only a subset E of O in the elimination phase is valuable. In fact, computing a shortest base (i.e., a subset I of minimal cardinality such that every variable not in I is definable in Σ in terms of I) [Lang and Marquis, 2008] would be prohibitive; indeed, computing such a base using a branch-and-bound algorithm would require, in the worst case, exponentially many definability tests in the number of variables occurring in Σ. Furthermore, while forgetting variables in Σ obviously leads to diminishing the number of variables occurring in it, it may also lead to an exponential increase of its size. Eliminating in Σ only a subset E of variables from those found in O renders it possible to focus on those variables for which the elimination step will not increase the size of Σ (à la NiVER [Subbarayan and Pradhan, 2004]), or only by a negligible factor. More generally, the elimination of an output variable can be committed only if the size of Σ after the elimination step remains small enough, once some additional preprocessing has been achieved. Among the equivalence-preserving preprocessings of interest are occurrence simplification [Lynce and Marques-Silva, 2003] and vivification [Piette *et al.*, 2008] (already considered in [Lagniez and Marquis, 2014]), which aim at shortening some clauses, and at removing some clauses (for vivification).

Example 2 (Example 1 cont'ed) *No literal equivalences, AND/OR gates or XOR gates are logical consequences of* Σ. *Nevertheless, since* Σ *implies*

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

a definability bipartition of $\mathit{Var}(\Sigma)$ *is* $\langle \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}, \{\mathrm{d}, \mathrm{e}\} \rangle$. *Now, forgetting* d *and* e *in* Σ *leads to the generation of two non-valid clauses* $\mathrm{a} \vee \mathrm{c}$ *and* $\mathrm{a} \vee \mathrm{b} \vee \mathrm{c}$ *so that* $\exists \{\mathrm{d}, \mathrm{e}\}.\Sigma$ *can then be computed as the conjunction of:*

$$\mathrm{a} \vee \mathrm{b}, \qquad \mathrm{a} \vee \mathrm{c}, \qquad \mathrm{a} \vee \mathrm{b} \vee \mathrm{c}.$$

which can be simplified further into $(\mathrm{a} \vee \mathrm{b}) \wedge (\mathrm{a} \vee \mathrm{c})$. *This* `CNF` *formula has only* 5 *models, hence this is also the case of* Σ.

Algorithm 2 shows how a bipartition $\langle \mathrm{I}, \mathrm{O} \rangle$ of $\mathit{Var}(\Sigma)$ 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), and initializes O with the corresponding variables (indeed, a literal ℓ belongs to the backbone of Σ precisely when $\mathit{var}(\ell)$ is defined in Σ in terms of $\emptyset$). Boolean Constraint Propagation is also done on Σ completed by its backbone

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 $\langle\Sigma, O\rangle \leftarrow$ `backbone`(Σ);
2 $V \leftarrow$ `sort`$(Var(\Sigma))$;
3 $I \leftarrow \emptyset$;
4 **foreach** $x \in V$ **do**
5   **if** `defined?`$(x, \Sigma, I \cup$ `succ`(x, V), `max#C`$)$ **then**
6         $O \leftarrow O \cup \{x\}$;
7   **else**
8         $I \leftarrow I \cup \{x\}$;
9 **return** O

(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 $\{I, O\}$ is a bipartition of $Var(\Sigma)$. At line 2, the remaining variables occurring in Σ are sorted by considering their number of occurrences from less to more frequent. At line 4, `defined?` takes advantage of Padoa's method (Theorem 2) for determining whether x is defined in Σ in terms of $I \cup$ `succ`(x, V), where `succ`(x, V) is the set of all variables of V which appear after x in V. `defined?` takes advantage of an anytime SAT solver `solve` based on CDCL architecture 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 $I \cup$ `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 which 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 which 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 $I \cup$ `succ`(x, V), while this could be questioned had a larger bound be considered). Clearly, the number of output variables found by B is not guaranteed to be maximal, but this is on purpose for the sake of efficiency (observe that the number of calls to `solve` does not exceed the number of variables occurring in Σ).

Algorithm 3 shows how variables from O are eliminated in Σ by E. P contains variables x from O which are candidates for elimination. P is initialized with the full set O (line 2). The main loop at line 3 is repeated 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 using a slight variant of the vivification algorithm `vivificationSimpl` reported in [Lagniez and Marquis, 2014]. Vivification [Piette *et al.*, 2008] is a preprocessing technique which aims at reducing the input CNF formula, i.e., to remove some clauses in it

Algorithm 3: E

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

1 $\Phi \leftarrow \Sigma$;
2 `iterate`$\leftarrow$true; $P \leftarrow O$;
3 **while** `iterate` **do**
4     $E \leftarrow P$; $P \leftarrow \emptyset$; `iterate`$\leftarrow$false;
5     $\Phi \leftarrow$ `vivificationSimpl`(Φ, E);
6   **while** $E \neq \emptyset$ **do**
7         $x \leftarrow$ `select`(E, Φ);
8         $E \leftarrow E \setminus \{x\}$;
9         $\Phi \leftarrow$ `occurrenceSimpl`(Φ, x);
10   **if** $\#(\Phi_x) \times \#(\Phi_{\neg x}) >$ `max#Res` **then**
11             $P \leftarrow P \cup \{x\}$
12   **else**
13             $R \leftarrow$ `removeSub`(`Res`$(x, \Phi), \Phi)$;
14   **if** $\#((\Phi \setminus \Phi_{x,\neg x}) \cup R) \leq \#(\Phi)$ **then**
15                 $\Phi \leftarrow (\Phi \setminus \Phi_{x,\neg x}) \cup R$;
16   `iterate`$\leftarrow$true;
17   **else**
18                 $P \leftarrow P \cup \{x\}$
19 **return** Φ

and some literals in the other clauses while preserving equivalence, using Boolean Constraint Propagation. 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, 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 a 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 [Lagniez and Marquis, 2014], 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 which 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, 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). Clearly, it can be the case that some variables of O are not eliminated by E, but again, this is on purpose for efficiency reasons.

4 Empirical Results

In our experiments, we have considered 703 CNF instances from the SAT LIBrary.[3] They are gathered into 8 data sets, as follows: BN (Bayesian networks) (192), BMC (Bounded Model Checking) (18), Circuit (41), Configuration (35), Handmade (58), Planning (248), Random (104), Qif (7) (Quantitative Information Flow analysis - security). Our experiments have been conducted on Intel Xeon E5-2643 (3.30 GHz) processors with 32 GiB RAM on Linux CentOS. A time-out of 1h and a memory-out of 7.6 GiB has been considered for each instance. We set max#Res to 500.

As a matter of comparison, we have considered the pmc preprocessor for model counting, described in [Lagniez and Marquis, 2014] and available from www.cril.fr/KC/. To be more precise, we considered pmc equipped with the *#eq* combination of preprocessings, 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 [Lagniez and Marquis, 2014].

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[4] [Sang *et al.*, 2004] and SharpSAT[5] [Thurley, 2006], 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 the C2D compiler [6] [Darwiche, 2001; 2004] for achieving the downstream model counting task. C2D generates a Decision-DNNF representation Σ^* of 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.

By the way, it is worth noting that B + E cannot be considered upstream to compilation-based approaches to model counting, while preserving the possibility of counting efficiently the number of models of the input conditioned by *any* consistent term. Indeed, when B + E(Σ) is not equivalent to Σ, the Decision-DNNF representation (B + E(Σ))* computed by C2D is not equivalent to Σ^*. Therefore, the possibility of efficient model counting after any conditioning is lost, but general conditioning must be downsized to a restricted form of conditioning where terms γ built up from I are allowed, but no other terms. Interestingly, such a restricted form of conditioning can prove enough in some scenarios. Especially, when the set of variables of Σ can be partitioned into a set of controllable variables (those which may require to be conditioned) and a remaining set of uncontrollable variables, one may take advantage of a slight variant of B + E ensuring that every controllable variable is put into I in order to simplify the input CNF formula Σ before compiling it.

Table 1 makes precise the number of instances (over 703) solved within 1h by each of the model counters Cachet, SharpSAT, and C2D (first column), when no preprocessing 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 preprocessing time is taken into account in the computations (it is part of the 1h CPU time allocated per instance).

model counter	no preprocessing	pmc	10	100	1000	∞
Cachet	525	558	586	588	594	602
SharpSAT	507	537	575	581	586	593
C2D	547	602	605	613	616	621

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

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

The cactus plot given in Figure 1 illustrates the performances of Cachet, SharpSAT, and C2D, 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 has been used). For the sake of readibility, only 10% of the dots have been printed. Again, the plot clearly shows B + E as a better preprocessor than pmc.

In order to determine how much applying B + E leads to reduction of the input CNF formula Σ compared to pmc, 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). Empirically, the results are presented on the two scatter plots (a) and (b) from Figure 2, where each point corresponds to an instance Σ, its x-coordinate corresponds to the value of the measure (#var (a) or #lit (b)) on pmc(Σ) equipped with the *#eq* combination of preprocessings, 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 ones. Clearly enough, B + E often

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

[4] www.cs.rochester.edu/~kautz/Cachet/

[5] sites.google.com/site/marcthurley/sharpsat

[6] reasoning.cs.ucla.edu/c2d/

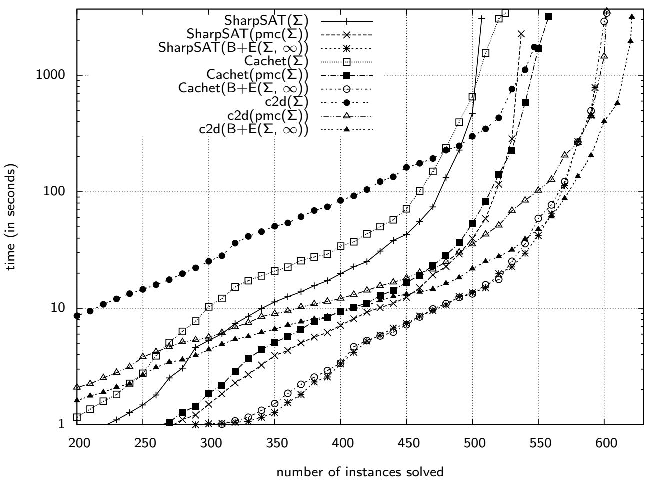


Figure 1: Comparing the performances of `Cachet`, `SharpSAT` and `C2D` depending on the preprocessing used (possibly none).

leads to much larger reductions than `pmc` for both measures. The benefits appear as very significant for instances from the Planning family.

Finally, we have evaluated how much $\texttt{B} + \texttt{E}$ leads to reduction of the overall model counting time compared to `pmc`. The results are presented on the two scatter plots (with logarithmic scales) (c) and (d) from Figure 2, for the two model counters `Cachet` and `C2D` (which appeared as the best counters in our experiments) considered downstream. Each point corresponds to an instance Σ, its x-coordinate corresponds to the time (in seconds) required to compute $\|\Sigma\|$ by computing $\texttt{pmc}(\Sigma)$ first, then calling the model counter on the resulting `CNF` formula, while its y-coordinate corresponds to the time required to compute $\|\Sigma\|$ by computing $\texttt{B} + \texttt{E}(\Sigma)$ (with $\texttt{max\#Conflicts}= \infty$) first, then calling the model counter on the resulting `CNF` formula. Again, whatever the downstream model counter, $\texttt{B} + \texttt{E}$ appears often as a more efficient preprocessor than `pmc`. The rightmost parts of the two scatter plots cohere with the results reported in Table 1, showing a number of instances which can be solved by any of the model counters when $\texttt{B} + \texttt{E}$ has been applied first, while they cannot be solved within the time limit of 1h when `pmc` is used instead. Finally, note that considering only the preprocessing times (and not the overall time needed to count the number of models of the input) for evaluating the preprocessor would be misleading: for some instances, the preprocessing times can be (relatively) long (details are available from `www.cril.fr/KC/`), just because the preprocessor 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).

5 Conclusion

We have defined a new preprocessing technique $\texttt{B} + \texttt{E}$ which associates with a given `CNF` formula Σ a `CNF` formula $\texttt{B} + \texttt{E}(\Sigma)$ which has the same number of models as Σ, but is often simpler w.r.t. the number of variables and the size. $\texttt{B} + \texttt{E}$ is based on standard theorems in classical logic by Beth and Padoa. Remarkably enough, while those results are quite old, they prove useful for defining a very effective preprocessing technique to model counting. Thus, experiments have shown that for many instances Σ, the overall computation time needed to calculate $\|\texttt{B} + \texttt{E}(\Sigma)\|$ using state-of-the art exact model counters is often much lower than the time needed to compute $\|\Sigma\|$ with the same counters.

This work opens a number of perspectives for further research. Considering other heuristics in `B` for determining a bipartition of the variables and determining how to tune the constants `max#C` and `max#Res` depending on the instance at hand will be studied in the future. Other perspectives concern the notion of projected model counting, as considered in [Aziz *et al.*, 2015]. The purpose is to compute $\|\exists E.\Sigma\|$ given a set E of variables and a formula Σ. Instead of taking advantage of $\texttt{B} + \texttt{E}$ followed by any model counter to compute $\|\Sigma\|$, we could instead use `B` followed by any projected model counter (where the projection is onto I). The other way around, we could also exploit `E` on E and Σ as a preprocessor for projected model counters. It would be interesting to implement both approaches and to determine whether they are helpful in practice.

Acknowledgments

We would like to thank the anonymous reviewers for their attentive reading, useful comments and advices.

References

[Apsel and Brafman, 2012] U. Apsel and R. I. Brafman. Lifted MEU by weighted model counting. In *Proc. of AAAI'12*, 2012.

[Audemard and Simon, 2009] G. Audemard and L. Simon. Predicting learnt clauses quality in modern sat solver. In *Proc. of IJCAI'09*, pages 399–404, 2009.

[Aziz *et al.*, 2015] R. A. Aziz, G. Chu, Ch. J. Muise, and P. J. Stuckey. #∃sat: Projected model counting. In *Proc. of SAT'15*, pages 121–137, 2015.

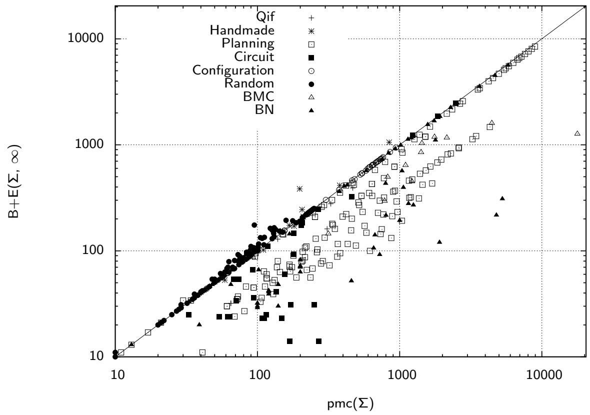
[Bacchus and Winter, 2004] F. Bacchus and J. Winter. Effective preprocessing with hyper-resolution and equality reduction. In *Proc. of SAT'04*, pages 341–355, 2004.

[Bacchus *et al.*, 2003] F. Bacchus, S. Dalmao, and T. Pitassi. Algorithms and complexity results for #sat and Bayesian inference. In *Proc. of FOCS'03*, pages 340–351, 2003.

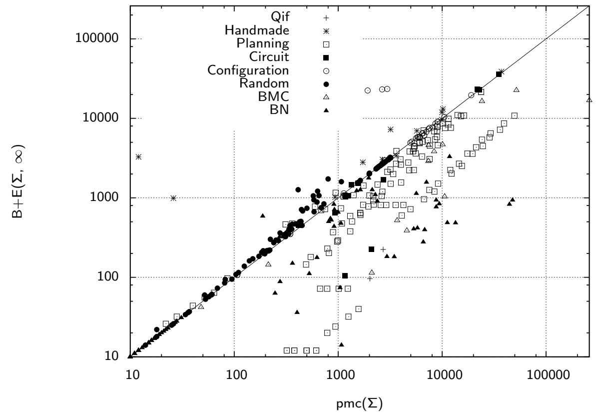
[Beth, 1953] E.W. Beth. On Padoa's method in the theory of definition. *Indagationes mathematicae*, 15:330–339, 1953.

[Biere, 2014] A. Biere. Lingeling essentials, A tutorial on design and implementation aspects of the the SAT solver lingeling. In *Proc. of POS'14*, page 88, 2014.

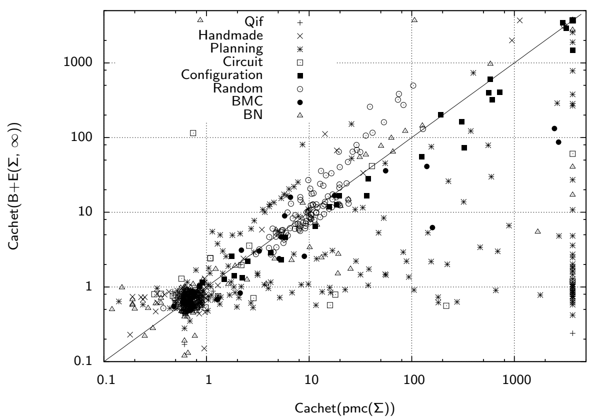
[Chavira and Darwiche, 2008] M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008.



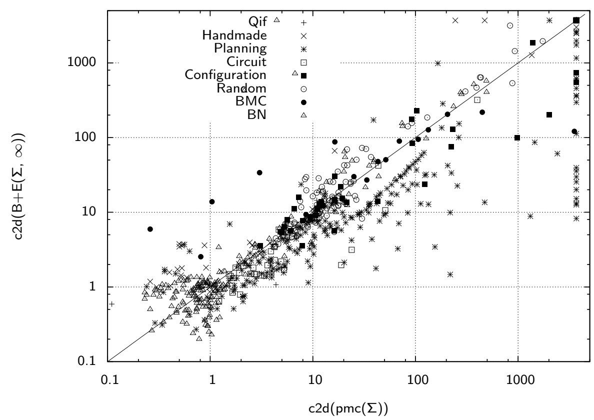
(a) #var



(b) #lit



(c) `B + E+Cachet` vs. `pmc+Cachet`



(d) `B + E+C2D` vs. `pmc+C2D`

Figure 2: #var reductions, #lit reductions, and model counting time reductions achieved by `pmc` vs. `B + E`

[Choi *et al.*, 2013] A. Choi, D. Kisa, and A. Darwiche. Compiling probabilistic graphical models using sentential decision diagrams. In *Proc. of ECSQARU'13*, pages 121–132, 2013.

[Darwiche, 2001] A. Darwiche. Decomposable negation normal form. *Journal of the Association for Computing Machinery*, 48(4):608–647, 2001.

[Darwiche, 2004] A. Darwiche. New advances in compiling cnf into decomposable negation normal form. In *Proc. of ECAI'04*, pages 328–332, 2004.

[Domshlak and Hoffmann, 2006] C. Domshlak and J. Hoffmann. Fast probabilistic planning through weighted model counting. In *Proc. of ICAPS'06*, pages 243–252, 2006.

[Een and Biere, 2005] N. Een and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proc. of SAT'05*, pages 61–75, 2005.

[Feiten *et al.*, 2012] L. Feiten, M. Sauer, T. Schubert, A. Czutro, E. Böhl, I. Polian, and B. Becker. #SAT-based vulnerability analysis of security components - A case study. In *Proc. of DFT'12*, pages 49–54, 2012.

[Gomes *et al.*, 2009] C. P. Gomes, A. Sabharwal, and B. Selman. Model counting. In *Handbook of Satisfiability*, pages 633–654. 2009.

[Han and Somenzi, 2007] H. Han and F. Somenzi. Alembic: An efficient algorithm for cnf preprocessing. In *Proc. of DAC'07*, pages 582–587, 2007.

[Heule *et al.*, 2010] M. Heule, M. Järvisalo, and A. Biere. Clause elimination procedures for cnf formulas. In *Proc. of LPAR'10*, pages 357–371, 2010.

[Heule *et al.*, 2011] M. Heule, M. Järvisalo, and A. Biere. Efficient cnf simplification based on binary implication graphs. In *Proc. of SAT'11*, pages 201–215, 2011.

[Järvisalo *et al.*, 2012] M. Järvisalo, A. Biere, and M. Heule. Simulating circuit-level simplifications on cnf. *Journal of Automated Reasoning*, 49(4):583–619, 2012.

[Lagniez and Marquis, 2014] J.-M. Lagniez and P. Marquis. Preprocessing for propositional model counting. In *Proc. of AAAI'14*, pages 2688–2694, 2014.

[Lang and Marquis, 2008] J. Lang and P. Marquis. On propositional definability. *Artificial Intelligence*, 172(8-9):991–1017, 2008.

[Lynce and Marques-Silva, 2003] I. Lynce and J. Marques-Silva. Probing-based preprocessing techniques for propositional satisfiability. In *Proc. of ICTAI'03*, pages 105–110, 2003.

[Manthey, 2012a] N. Manthey. Coprocessor 2.0 - A flexible CNF simplifier - (tool presentation). In *Proc. of SAT'12*, pages 436–441, 2012.

[Manthey, 2012b] N. Manthey. Solver description of RISS 2.0 and PRISS 2.0. Technical report, TU Dresden, Knowledge Representation and Reasoning, 2012.

[Padoa, 1903] 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*, pages 309–365. Paris, 1903.

[Palacios *et al.*, 2005] H. Palacios, B. Bonet, A. Darwiche, and H. Geffner. Pruning conformant plans by counting models on compiled d-DNNF representations. In *Proc. of ICAPS'05*, pages 141–150, 2005.

[Piette *et al.*, 2008] C. Piette, Y. Hamadi, and L. Saïs. Vivifying propositional clausal formulae. In *Proc. of ECAI'08*, pages 525–529, 2008.

[Samer and Szeider, 2010] M. Samer and S. Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010.

[Sang *et al.*, 2004] T. Sang, F. Bacchus, P. Beame, H.A. Kautz, and T. Pitassi. Combining component caching and clause learning for effective model counting. In *Proc. of SAT'04*, 2004.

[Sang *et al.*, 2005] T. Sang, P. Beame, and H. A. Kautz. Performing Bayesian inference by weighted model counting. In *Proc. of AAAI'05*, pages 475–482, 2005.

[Subbarayan and Pradhan, 2004] S. Subbarayan and D.K. Pradhan. NiVER: Non increasing variable elimination resolution for preprocessing SAT instances. In *Proc. of SAT'04*, pages 276–291, 2004.

[Thurley, 2006] M. Thurley. sharpSAT - counting models with advanced component caching and implicit BCP. In *Proc. of SAT'06*, pages 424–429, 2006.

[Tseitin, 1968] G.S. Tseitin. *On the complexity of derivation in propositional calculus*, chapter Structures in Constructive Mathematics and Mathematical Logic, pages 115–125. Steklov Mathematical Institute, 1968.