Dualizing Projected Model Counting

Sibylle Möhle
Johannes Kepler University Linz
Linz, Austria

Armin Biere
Johannes Kepler University Linz
Linz, Austria

***Abstract*—In many recent applications of model counting not all variables are relevant for a specific problem. For instance redundant variables are added during formula transformation. In projected model counting these redundant variables are ignored by projecting models onto relevant variables. Inspired by dual propagation which has its origin in solving quantified Boolean formulae and jointly works on both the original formula and its negation, we present a novel calculus for dual projected model counting. It allows to capture existing techniques such as blocking clauses, chronological as well as non-chronological backtracking, but also introduces new concepts including discounting and dual conflict analysis to obtain partial models. Experiments demonstrate the benefit of our approach.**

I. Introduction

Classical applications of #SAT, the task of counting the models of a propositional theory, are found in the area of probabilistic reasoning [1] adopted in, e.g., medical diagnosis and planning. Further application scopes encompass circuit design [2] and quantitative information flow analysis [3] as well as differential cryptanalysis [4]. In product configuration, the number of valid configurations under given preconditions might be of interest [5]. For some tasks not all configuration options might be relevant, thus only the models projected onto the relevant variables are counted [6]. Projected models may be required in planning as well [7], [8]. The task of *projected model counting* is also referred to as #∃SAT, since the main idea of projection is to existentially quantify the irrelevant variables, i.e., the variables which are to be discarded. In this sense, #SAT is a special case of #∃SAT, namely the one in which the set of irrelevant variables is empty [8].

Particular challenges in model counting. In propositional model counting, contrarily to SAT solving, the search does not terminate after the detection of a model. Instead, the search space needs to be explored exhaustively. Conflict Driven Clause Learning (CDCL) [9], [10] provides efficient means to deal with conflicting assignments, but a comparable method for handling satisfying assignments is still not available. Some state-of-the-art #SAT solvers prune the search space upon finding a satisfying assignment by adding blocking clauses with the aim to prevent multiple model counts [11]. An apparent drawback of this approach is a substantial growth of the formula since these blocking clauses are irredundant and therefore must not be deleted. This issue is addressed in [12], a solver for projected model enumeration working without blocking clauses, and in [13] where blocking clauses are eagerly deleted and the number of kept blocking clauses is at any time limited to be at most linear in the number of relevant variables. Although devised for projected answer set enumeration, this method is readily applicable for #∃SAT. The addition of large clauses may furthermore slow down the solver. On this account, in #CLASP [7] the blocking clauses are minimized which on the one hand reduces their count and on the other hand prunes a larger portion of the search space. To detect partial models, the DPLL-based model counter CDP [14] performs satisfiability checks on the residual of the formula. This is not the case in state-of-the-art SAT solvers which only keep track of the assigned variables. From a complexity point of view this makes sense in SAT, since a satisfying assignment can always be extended to a total model by a number of decisions linear in the variable count and thus compensates for the overhead introduced by, e.g., clause watching. In contrast, detecting partial models and subsequent pruning of the search space leads to a higher performance gain in #SAT, hence more computational effort may be invested. For the same reason, more involved preprocessing techniques are justified in model counting [15].

Our motivation. Inspired by [16], Abstract Dual #DPLL [17] was developed with the aim to investigate the suitability of a dual approach for exact model counting as an alternative to component analysis [18]–[20] which can be considered the state of the art in exact #SAT solving. One objective of our work presented here is to extend this approach to facilitate projected model counting. A second goal is to incorporate methods which are widely used in state-of-the-art SAT solvers. Our third goal is to investigate the impact of a dual approach on the performance of a non-dual method. We only partially consider preprocessing [15], [21] in this work, but plan to further investigate its applicability in a dual setting and to develop adequate methods in the future.

Our contribution. In this work we present the first dual calculus addressing projected exact unweighted model counting. First, we present a dual representation of the formula under consideration which facilitates the detection of partial models and subsequent pruning of the search space. Our method incorporates "good learning" and is exempt from satisfiability checks and clause watching mechanisms. This results in a significant performance gain compared to the non-dual variant of the same algorithm which finds only total models. Second, our calculus takes an arbitrary formula or a circuit as argument and returns the correct model count even if for its transformation into CNF techniques are adopted

Supported by Austrian Science Fund (FWF) grant S11408-N23 (RiSE).

which in general do not preserve the model count, such as the Plaisted-Greenbaum transformation [22]. Third, we introduce a novel technique based on *flipping* and *discounting* which prevents multiple model counts without the use of blocking clauses. Our calculus models techniques implemented in state-of-the-art SAT solvers, such as conflict analysis and conflict-driven backjumping. Finally, we provide a robust and carefully tested implementation DUALIZA of our calculus.

In Figure 1 we provide a simple example to highlight the power of dual projected model counting. More details are discussed in Section VI.

The paper is organized as follows: In Section II, we discuss basic concepts. Section III introduces the concept of duality and our new duality property. Our calculus is presented in Section IV on which our implementation described in Section V is based. After experiments in Section VI, we discuss related work in Section VII before we conclude in Section VIII.

II. PRELIMINARIES

Let F be an arbitrary (propositional) formula over variables Z, interpreted over Boolean constants $\mathbb{B} = \{0, 1\}$. Further assume Z to be partitioned into the set of *relevant input variables* X and the set of *irrelevant input variables* Y. We denote with *inputs* the variables contained in either X or Y. A total assignment $\sigma\colon Z \to \mathbb{B}$ maps Z to the truth values 1 and 0 and can be applied to F to yield a truth value $\sigma(F) \in \mathbb{B}$, also written $F|_\sigma$. A *relevant input assignment* is defined on X and undefined elsewhere. Similarly, an *irrelevant input assignment* is defined. This lets us decompose any total $\sigma = \sigma_X \cup \sigma_Y$ into its relevant σ_X and irrelevant σ_Y part. We use $F(U)$, or equivalently $\mathrm{var}(F) \subseteq Z$, to denote that F only depends on (contains) a subset $U \subseteq Z$ of all variables and write $F(X, Y)$ if F is defined over $X \cup Y$.

We count the number of satisfying assignments of $F(X, Y)$ projected onto the relevant variables X, defined as

$$\#\exists Y.\, F(X,Y) = |\{\tau\colon X \to \mathbb{B} \mid \text{exists } \sigma : Z \to \mathbb{B} \text{ with } \sigma\big(F(X,Y)\big) = 1 \text{ and } \tau = \sigma_X\}|.$$

Thus #SAT turns out to be the special case where $Y = \emptyset$.

In order to make use of sophisticated data structures and algorithms used in modern SAT solvers we further consider propositional formulae in CNF and thus for instance apply Tseitin transformation [23] on F to obtain a CNF representation $P(X, Y, S)$ of F depending on X and Y as well as on variables S introduced by the transformation. Note that, even though the Tseitin encoding is only satisfiability-preserving, i.e., the result is not logically equivalent, it does preserve the number of models, assuming all inputs are relevant. The models of F projected onto X are exactly the models of its Tseitin representation $\exists S.P(X, Y, S)$ projected onto X, and

$$\#\exists Y, S.\, P(X,Y,S) = \#\exists Y.\, F(X,Y).$$

Our approach uses two formulae to capture the projected model counting problem. The *primal* formula $P(X, Y, S)$ ranges over the inputs X and Y and the *primal* variables S, while the *dual* formula $N(X, Y, T)$ ranges over the same inputs X and Y but also over the *dual* variables T instead of the primal variables S. The idea is that N is the "negation" of F, and hence of P, which is easy to achieve by encoding the negation of F using Tseitin variables. The precise condition is discussed further down. In line with the definition of an assignment, a *primal assignment* σ_S and a *dual assignment* σ_T are defined. This extends our notion of total assignment to $\sigma = \sigma_X \cup \sigma_Y \cup \sigma_S \cup \sigma_T$ over variables $V = X \cup Y \cup S \cup T$.

A CNF F over V is a conjunction of clauses and each clause is a disjunction of literals. A literal ℓ is either a variable $v \in V$ or its negation $\neg v$. In both cases we write $\mathrm{var}(\ell) = v$ and extend this notion to sequences and sets of literals as well as formulae. We also use $\bar{\ell} = \neg\ell$ and assume $\neg\neg\ell = \ell$. We also write $C \in F$ if C is a clause of F and similarly $\ell \in C$. A sequence $I = \ell_1 \cdots \ell_n$ of literals with mutually exclusive variables ($\mathrm{var}(\ell_i) \neq \mathrm{var}(\ell_j)$ for $i \neq j$) is called a *trail*. Trails can be concatenated, $I = I'I''$, assuming the variables in I' and I'' are distinct. We also use $\ell \in I$, treating I as set of literals, and in particular define $X - I = X - \mathrm{var}(I) \subseteq X$, the subset of variables of X not in I. These trails are also interpreted as partial assignments with $I(\ell) = 1$ iff $\ell \in I$. Thus $I(\ell) = 0$ if $\bar{\ell} \in I$ and $I(\ell)$ is undefined if $\mathrm{var}(\ell) \notin \mathrm{var}(I)$. This gives the useful shortcut $2^{|X-I|}$ to denote the number of (total) relevant input assignments covered by the partial assignment represented by the trail I. We denote the projection of I onto X by $\pi(I, X)$ and consider it also a conjunction of literals. The *residual* of F with respect to I, denoted by $F|_I$, is obtained by replacing in F the literals occurring in I by their truth value. I *satisfies* F, denoted by $I \models F$, if $F|_I = \emptyset$. Trail I *falsifies* F if it contains the empty clause, i.e., $\emptyset \in F|_I$.

We are going to present a proof calculus in the style of DPLL(T) [24] and will also include elements of CDCL formalized in [25]. Proof rules model state transitions of an abstract projected model counting DPLL / CDCL solver.

Besides primal and dual formulae, the main ingredient of an abstract state is a trail which, as discussed above, captures the current partial assignment and in addition marks some of its literals ℓ as decision literals, using the notation ℓ^d. These denote open "left" (or "first") branches in the sense of DPLL.

We also mark literals ℓ starting a "right" (or "second") branch with an f followed by the number of models found at the corresponding decision level in parenthesis, i.e., $\ell^{f(m)}$. The idea is that after closing the left branch of a decision its decision literal ℓ^d becomes a *flipped literal* $\bar{\ell}^{f(m)}$ starting the right branch of that decision maintaining its decision level $\mathrm{dl}(\ell)$. We denote the set of decision literals in I by $\mathrm{decs}(I)$ and interpret it as a conjunction of literals where appropriate. In an analogous manner, we represent the set of flipped literals in I by $\mathrm{flips}(I)$ and the set of unit literals in a residual $G|_I$ of a formula G w.r.t. I by $\mathrm{units}(G|_I)$.

Backjumping involves the addition of redundant clauses which may be deleted anytime. We mark redundant clauses with an r, writing C^r, to distinguish them from blocking clauses which prevent multiple model counts and therefore have to be retained until the end of the procedure.

```
$ cat clause.form
a | b | c | d
$ dualiza -e clause.form
ALL SATISFYING ASSIGNMENTS
d
c !d
b !c !d
a !b !c !d
$ dualiza clause.form
NUMBER SATISFYING ASSIGNMENTS
15
```

```
$ dualiza clause.form -l | grep RULE
c LOG 1 RULE UNX 1     -4
c LOG 1 RULE UNX 2     -4
c LOG 1 RULE BN0F 1    -4
c LOG 2 RULE UNX 3     -3
c LOG 2 RULE BN0F 2    -3
c LOG 3 RULE UNX 4     -2
c LOG 3 RULE BN0F 3    -2
c LOG 3 RULE UP 1      1
c LOG 3 RULE EP1 1
```

Fig. 1. On the left hand side of the figure our implementation DUALIZA is applied to a simple example consisting of a single clause. The "log" output on the right hand side demonstrates that in dual mode this example is solved in essence by just dual propagation (UNX) (see Section VI for details).

III. DUALITY

In propositional model counting the search space needs to be processed exhaustively. In this sense #SAT exhibits a certain analogy to QBF solving where, due to the existence of universal quantifiers in the formula, the complete search space needs to be traversed.

To overcome this limitation, dual propagation was introduced for circuits in [26] and adapted to CNF-based QBF solving in [27]. This work inspired our abstract framework developed with focus on DPLL [17]. In the work reported here, we extend this approach in two ways: first by projection and second by elements of CDCL.

Let $F(X,Y)$ be an arbitrary (propositional) formula and our task is to compute the number of models of F projected onto X. Following the concepts introduced in Section II, we compute a *dual representation* of F consisting of two CNF formulae $P(X,Y,S)$ and $N(X,Y,T)$. Recall that P and N encode F and its negation, respectively.

Our model counter executes a dual variant of CDCL on P and N simultaneously and maintains a single trail I with $\mathrm{var}(I) = X \cup Y \cup S \cup T$. The input variables in X and Y are shared by P and N and are called *shared variables*. They may be propagated in either formula [26].

Basic idea. Every trail I either satisfying P or falsifying N is a (partial) model of F and represents $2^{|X-I|}$ total models of F projected onto X. In contrast, no model of F can be computed if I falsifies P. Obviously, the same holds if I satisfies N, but due to the structure of I this situation will never arise in our framework as will be clarified further down. Conflict analysis is performed after a conflict is detected, and backtracking occurs upon either a conflict or the detection of a model of F. If I is a partial assignment, backtracking prunes a potentially large portion of the search space. The procedure terminates as soon as the complete search space has been processed.

Thus in our approach we assume $F(X,Y)$ is represented by a pair of dual formulae satisfying the following definition.

Definition 1 (Combined Formula Pair): A *combined formula pair* of a formula $F(X,Y)$ consists of formulae $P(X,Y,S)$ and $N(X,Y,T)$ meeting the following conditions:

$$\exists S.\, P(X,Y,S) \equiv \quad F(X,Y) \tag{1}$$

$$\exists T.\, N(X,Y,T) \equiv \neg F(X,Y) \tag{2}$$

where X, Y, S and T are pairwise disjoint sets of variables. We denote a combined formula pair of $F(X,Y)$ by $[P(X,Y,S) \parallel N(X,Y,T)](F)$ or $[P \parallel N](F)$.

Definition 1 essentially states that F and P are semantically equivalent, i.e., have the same models, upon projection onto $X \cup Y$ and that the same holds for $\neg F$ and N.

As a consequence, if P and N are a combined formula pair of F, for every total assignment σ of X and Y it holds that $\sigma(\exists S.P(X,Y,S)) \neq \sigma(\exists T.N(X,Y,T))$.

Definition 2 (Duality Property): Let X, Y, S and T be pairwise disjoint sets of variables. Two formulae $G(X,Y,S)$ and $H(X,Y,T)$ comply with the *duality property*, if

$$\forall X,Y.\, ((\exists\, S.\, G(X,Y,S)) \oplus (\exists\, T.\, H(X,Y,T))) \tag{3}$$

where "$\oplus$" denotes "exclusive or (XOR)".

Lemma 1: The duality property holds for a combined formula pair $[P(X,Y,S) \parallel N(X,Y,T)](F)$.

Consider an assignment I (trail) with $P(X,Y,S)|_{I'} \models I$ for $I' = \pi(I, X \cup Y)$, which for instance holds if variables in X and Y are the only decisions in I and the rest of I is obtained through unit propagation in P. Then the dual property in (3) continues to hold for residuals w.r.t. I:

$$\forall X,Y.\, ((\exists\, S.\, P(X,Y,S)|_I) \oplus (\exists\, T.\, N(X,Y,T)|_I)) \tag{4}$$

This property provides the most important invariant for our calculus discussed in the next section, but requires that we split on X and Y variables first.

In general, we assume that only (3) holds for P and N, without necessarily requiring that there exists an F satisfying Definition 1. In this situation, even after assigning X and Y and performing unit propagation, there might still be an unassigned variable $s \in S$ left. Assume we extend the trail with the decision ℓ with $\mathrm{var}(\ell) = s$. At this point the dual property (4) might seize to hold, since the value picked for s may lead to an unsatisfiable residual $P(X,Y,S)|_{I\ell}$ even if $P(X,Y,S)|_{I\bar{\ell}}$ is satisfiable, thus both $\exists\, S.\, P(X,Y,S)|_{I\ell}$ and $\exists\, T.\, N(X,Y,T)|_{I\ell}$ are false.

For correctness it is sufficient to first split on relevant variables X, followed by irrelevant variables Y and primal variables S, but never split on dual variables T. This splitting order maintains the following direction (5) of the dual property (4) on residuals, namely that a conflict in N guarantees that all extensions to the current assignment to relevant variables in X can be extended to total models of P:

$$\forall X, Y.\, ((\neg\exists\, T.\, N(X,Y,T)|_I) \rightarrow (\exists\, S.\, P(X,Y,S)|_I)) \quad (5)$$

A formal proof of this invariant and accordingly the correctness of our calculus is out of scope of this paper. For now we rely on extensive testing and thus an empirical justification.

IV. Calculus

We describe the framework of our calculus by a labeled state transition system $\langle S, L, \rightsquigarrow, s_0\rangle$ with set of states S, set of labels L and transition relation $\rightsquigarrow \subseteq S \times S$. Intermediate states are of the form (P, N, I, M) where $P(X,Y,S)$ and $N(X,Y,T)$ are a combined formula pair of $F(X,Y)$, I is a trail with $\mathrm{var}(I) \in X \cup Y \cup S \cup T$ and $M \in \mathbb{N}$. The initial state is defined by $s_0 = (P, N, (), 0)$ with $()$ denoting the empty trail. The terminal state is $M \in \mathbb{N}$ representing $\#\exists Y.F(X,Y)$. The transition relation $\rightsquigarrow$ is defined as the union of transition relations $\rightsquigarrow_l$ with $l \in L$. The labels indicate the action taken (1st letter), to which formula or variable set it is applied (2nd letter), whether it is triggered by a satisfying or falsifying assignment (3rd letter) and whether a blocking clause is learned or flipping is applied (4th letter). The rules are listed in Figure 2.

Terminate search. The search terminates as soon as I either satisfies or falsifies either P or N and the relevant search space has been traversed exhaustively. If I falsifies P, $\pi(I,X)$ can not be extended to a model of $\exists Y.F(X,Y)$, and M remains unaltered (EP0). Requiring that no decision literals are left on the trail ensures that no models are missed due to a "wrong" assignment of variables in S. If I either satisfies P or falsifies N, $\pi(I,X)$ can be extended to $2^{|X-I|}$ models of $\exists Y.F(X,Y)$, and M is updated accordingly (EP1 and EN0). Requiring that no relevant decision literal is left on the trail is sufficient since in the presence of irrelevant or primal decision literals all relevant variables are assigned. Flipping an irrelevant or primal decision literal therefore would yield redundant models w.r.t. projection onto X.

Backtracking. If the partial interpretation represented by the trail falsifies P, the solver may backtrack chronologically and turn the last decision literal ℓ^d into a flipped decision literal $\overline{\ell}^{f(m)}$ where m equals the number of models detected at decision level $> \mathrm{dl}(\ell)$. This model count is obtained by summing up the model counts assigned to the flipped decision literals with decision level higher than $\mathrm{dl}(\ell)$, while M remains unaltered (BP0F). Alternatively, a redundant clause may be learned which becomes unit for I, the solver backtracks non-chronologically and propagates this new unit literal. Backjumping involves discarding the models found in I', hence their count is subtracted from M to prevent multiple counts when they are found again (JP0). If the partial interpretation represented by the trail falsifies N, its projection onto X can be extended to a model of F, and M is incremented by the number of total models projected onto X represented by the trail. Flipping the last irrelevant or primal decision literal would yield redundant models w.r.t. projection onto X. Therefore, when backtracking chronologically, the solver turns the last relevant decision literal into a flipped decision literal and assigns it the sum of the number of models represented by the actual trail projected onto X and all models detected at decision levels $> \mathrm{dl}(\ell)$ (BN0F). Alternatively, a blocking clause is added to P (BN0L). If Il^dI' satisfies P, $\pi(I\ell I', X)$ can be extended to $m'' = 2^{|X-I\ell I'|}$ models of $\exists Y.F(X,Y)$, and M is incremented by m''. As discussed above, the last relevant decision literal ℓ must be considered. It may be turned into a flipped decision literal and assigned the sum of m'' and all number of models detected at decision level $> \mathrm{dl}(\ell)$ (BP1F). Alternatively, a blocking clause may be added to P (BP1L).

Decisions. $P|_I$ and $N|_I$ contain neither a unit nor the empty clause. Relevant input variables are prioritized (DX) over irrelevant input and primal variables (DYS). Assigning primal variables before irrelevant input variables might result in a conflict in P but has no effect on N and hence does not affect the model count. In this case, the duality property might not hold for the residuals as discussed in Section III.

Unit propagation. Literals are propagated in both P and N. Unit propagation in P is prioritized over unit propagation in N and is executed as in SAT (UP). For unit propagation in N, two cases need to be considered. If $\mathrm{var}(\ell) \in X \cup Y$, I is extended by $\overline{\ell}^d$ to enforce backtracking due to a conflict in $N|_{I\overline{\ell}}$ in the next step (UNXY). Otherwise, it is propagated (UNT).

Forgetting redundant clauses. Deletion of redundant clauses is equivalence-preserving, hence redundant clauses may be removed anytime (FP) assuming P is conflict-free under I.

Blocking clauses in dual mode. Blocking clauses shall impede the multiple detection of models. Since they are added exclusively to P, this fails in the dual setting if a trail falsifies N. Consider $F(X,\emptyset) = (\overline{1} \vee 2) \wedge (1 \vee 2)$ over $X = \{1,2,3,4\}$ and $Y = \emptyset$. We define $P = F$ and $N = (\overline{5} \vee 1) \wedge (\overline{5} \vee \overline{2}) \wedge (\overline{6} \vee \overline{1}) \wedge (\overline{6} \vee \overline{2}) \wedge (5 \vee 6)$, hence $S = \emptyset$ and $T = \{5,6\}$. After deciding 4, 3 and 2 (DX), $\overline{5}$ and $\overline{6}$ are propagated in N (UNT) resulting in a conflict in $N|_I$ with $I = (4^d, 3^d, 2^d, \overline{5}, \overline{6})$ and $\pi(I,X) = (4,3,2)$. The solver adds the blocking clause $(\overline{2} \vee \overline{3} \vee \overline{4})$ to P, sets $M = 2$ and backtracks chronologically (BN0L). After propagating $\overline{1}$ (UP), P is falsified. JP0 is applied yielding the redundant unit clause (2), and the solver jumps back to decision level 0. It propagates 2 (UP), $\overline{5}$ and $\overline{6}$ (UNT) resulting in $I = (2, \overline{5}, \overline{6})$ which falsifies N. The partial model $\pi(I,X) = (2)$ represents $2^3 = 8$ total models of F. Note that this model subsumes the one found previously. **In our framework, we therefore have to disallow the combination of blocking clauses and dual reasoning.** This means that "L" rules can not be combined with "N" rules. Thus only "F" rules are allowed if we insist on using "N" rules, which is the default in our implementation.

A workaround would be the following: If a blocking clause is added to P, either its negation is added disjunctively to N

EP0: $(P, N, I, M) \rightsquigarrow_{\mathsf{EP0}} M$ if $\emptyset \in P|_I$ and $\mathrm{decs}(I) = \emptyset$

EP1: $(P, N, I, M) \rightsquigarrow_{\mathsf{EP1}} M + 2^{|X-I|}$ if $P|_I = \emptyset$ and $\mathrm{var}(\mathrm{decs}(I)) \cap X = \emptyset$

EN0: $(P, N, I, M) \rightsquigarrow_{\mathsf{EN0}} M + 2^{|X-I|}$ if $\emptyset \in N|_I$ and $\mathrm{var}(\mathrm{decs}(I)) \cap X = \emptyset$

BP0F: $(P, N, I\ell^d I', M) \rightsquigarrow_{\mathsf{BP0F}} (P, N, I\bar{\ell}^{f(m')}, M)$ if $\emptyset \in P|_{I\ell I'}$ and $\mathrm{var}(\mathrm{decs}(I')) = \emptyset$ and $m' = \sum \{m \mid \ell^{f(m)} \in I'\}$

JP0: $(P, N, II', M) \rightsquigarrow_{\mathsf{JP0}} (P \wedge C^r, N, I\ell', M - m')$ if $\emptyset \in P|_{II'}$ and $P \models C$ and $C|_I = \{\ell'\}$ and $m' = \sum \{m \mid \ell^{f(m)} \in I'\}$

BN0F: $(P, N, I\ell^d I', M) \rightsquigarrow_{\mathsf{BN0F}} (P, N, I\bar{\ell}^{f(m'+m'')}, M + m'')$ if $\emptyset \in N|_{I\ell I'}$ and $\mathrm{var}(\ell) \in X$ and $\mathrm{var}(\mathrm{decs}(I')) \cap X = \emptyset$ and $m' = \sum \{m \mid \ell^{f(m)} \in I'\}$ and $m'' = 2^{|X-I\ell I'|}$

BN0L: $(P, N, I\ell^d I', M) \rightsquigarrow_{\mathsf{BN0L}} (P \wedge D, N, I\bar{\ell}, M + m'')$ if $\emptyset \in N|_{I\ell I'}$ and $\mathrm{var}(\ell) \in X$ and $\mathrm{var}(\mathrm{decs}(I')) \cap X = \emptyset$ and $m'' = 2^{|X-I\ell I'|}$ and $D = \pi(\neg\mathrm{decs}(I\ell), X)$

BP1F: $(P, N, I\ell^d I', M) \rightsquigarrow_{\mathsf{BP1F}} (P, N, I\bar{\ell}^{f(m'+m'')}, M + m'')$ if $P|_{I\ell I'} = \emptyset$ and $\mathrm{var}(\ell) \in X$ and $\mathrm{var}(\mathrm{decs}(I')) \cap X = \emptyset$ and $m' = \sum \{m \mid \ell^{f(m)} \in I'\}$ and $m'' = 2^{|X-I\ell I'|}$

BP1L: $(P, N, I\ell^d I', M) \rightsquigarrow_{\mathsf{BP1L}} (P \wedge D, N, I\bar{\ell}, M + m'')$ if $P|_{I\ell I'} = \emptyset$ and $\mathrm{var}(\ell) \in X$ and $\mathrm{var}(\mathrm{decs}(I')) \cap X = \emptyset$ and $m'' = 2^{|X-I\ell I'|}$ and $D = \pi(\neg\mathrm{decs}(I\ell), X)$

DX: $(P, N, I, M) \rightsquigarrow_{\mathsf{DX}} (P, N, I\ell^d, M)$ if $\emptyset \notin (P \wedge N)|_I$ and $\mathrm{units}((P \wedge N)|_I) = \emptyset$ and $\mathrm{var}(\ell) \in X - I$

DYS: $(P, N, I, M) \rightsquigarrow_{\mathsf{DYS}} (P, N, I\ell^d, M)$ if $\emptyset \notin (P \wedge N)|_I$ and $\mathrm{units}((P \wedge N)|_I) = \emptyset$ and $\mathrm{var}(\ell) \in (Y \cup S) - I$ and $X - I = \emptyset$

UP: $(P, N, I, M) \rightsquigarrow_{\mathsf{UP}} (P, N, I\ell, M)$ if $\{\ell\} \in P|_I$

UNXY: $(P, N, I, M) \rightsquigarrow_{\mathsf{UNXY}} (P, N, I\bar{\ell}^d, M)$ if $\{\ell\} \in N|_I$ and $\mathrm{var}(\ell) \in X \cup Y$ and $\emptyset \notin P|_I$ and $\mathrm{units}(P|_I) = \emptyset$

UNT: $(P, N, I, M) \rightsquigarrow_{\mathsf{UNT}} (P, N, I\ell, M)$ if $\{\ell\} \in N|_I$ and $\mathrm{var}(\ell) \in T$ and $\emptyset \notin P|_I$ and $\mathrm{units}(P|_I) = \emptyset$

FP: $(P \wedge C^r, N, I, M) \rightsquigarrow_{\mathsf{FP}} (P, N, I, M)$ if $\emptyset \notin P|_I$

Fig. 2. The complete set of rules of our framework, where $P(X, Y, S)$ and $N(X, Y, T)$ form a combined formula pair of $F(X, Y)$, and I denotes the trail over variables $X \cup Y \cup S \cup T$. The rules cover termination (rule name starting with E), chronological and non-chronological backtracking (B and J), decisions (D) and unit propagation (U) as well as clause forgetting (F). They may be applied either to P or N (P or N) and triggered by a falsifying (0) or satisfying (1) assignment. Letters X, Y, S and T denote the variable sets the rules are applied to (X, Y, S and T). Finally, either a blocking clause may be learned (L) or flipping is applied (F). Blocking clauses are added to P only and thus fail to prevent multiple model counts if a model is found due to a conflict in N. We therefore disallow the combination of blocking clauses and dual reasoning. Discounted models might not be detected again if they are subsumed by a blocking clause recorded previously. We therefore also disallow the combination of discounting and blocking clauses. For more details including examples we refer to paragraphs *Blocking clauses in dual mode* and *Combining blocking clauses and discounting* in Section IV.

or, whenever a conflict in N occurs, it must be ensured that none of the blocking clauses is falsified by the current trail. In the first case, N is not a CNF anymore, and the rules and preconditions involving N and the whole search procedure need to be adapted accordingly. In the second case we need to keep track of the blocking clauses instead, e.g., in a CNF R (*refutations*), and check whether R is satisfied prior to count a model whenever N is falsified by I.

Combining blocking clauses and discounting. If previously found models are *discounted* upon backjumping and learning a blocking clause, they may be lost. The problem in this case arises if these models are blocked by the learned clause. Consider $F = (1 \vee \overline{2}) \wedge (1 \vee \overline{3}) \wedge (\overline{1} \vee \overline{2})$. $P = F$ and $N = (\overline{6} \vee \overline{1}) \wedge (\overline{6} \vee 2) \wedge (\overline{7} \vee \overline{1}) \wedge (\overline{7} \vee 3) \wedge (\overline{8} \vee 1) \wedge (\overline{8} \vee 2) \wedge (6 \vee 7 \vee 8)$ with $X = \{1, 2, 3, 4, 5\}$, $Y = \emptyset$, $S = \emptyset$, and $T = \{6, 7, 8\}$. After deciding 5, 4, and 3, 1 and $\overline{2}$ are propagated which leads to the detection of model $m_1 = (1, \overline{2}, 3, 4, 5)$. The last decision is flipped (BP1F) and after propagation $\overline{7}$, deciding $\overline{2}$ and propagating $\overline{6}$ and $\overline{8}$, a second model $m_2 = (\overline{2}, \overline{3}, 4, 5)$ is found. The solver backtracks and flips the last decision (BN0F). At a later stage, BP0F is executed, and $I = (5^d, \overline{4}^{f(3)})$. The model count associated with the flipped decision literal $\overline{4}^{f(3)}$ refers to models m_1 and m_2. Later on, a third model $m_3 = (1, \overline{2}, \overline{4}, 5)$ is detected and (BN0F) executed. After one more propagation step, model $m_4 = (\overline{1}, \overline{2}, \overline{3}, \overline{4}, 5)$ is detected and a blocking clause $b = (2 \vee \overline{5})$ learned (BP1L). Due to the application of JP0 in the further procedure, the solver jumps back to level 0, i.e., past the flipped decision literal $\overline{4}^{f(3)}$. It discounts models m_1 and m_2, i.e., 3 total models. Since these models are blocked by b, they can not be found again. In fact, during the further execution the models $m_5 = (1, \overline{2}, \overline{5})$ and $m_6 = (\overline{1}, \overline{2}, \overline{3}, \overline{5})$ are found before terminating with EN0, and the solver returns $M = 9$ instead of $M = 12$. Combining the use of blocking clauses with discounting therefore conditions to discount only models which are not blocked by the learned clause and to add blocking clauses for these models as soon as the respective flipped decision literal is removed from I. An obvious drawback of this method is that these checks are expensive since the number of models may be exponential in the number of relevant variables. **Thus our framework also disallows the combination of discounting and blocking clauses**. This means that as soon we have discounting enabled, actually backjumping (JP0) with $m' \neq 0$, we can not use blocking clauses ("L" rules). This provides another reason to disable "L" rules in our implementation by default, even though for testing purposes blocking clauses can be enabled (if dual reasoning and discounting are disabled).

The demonstrated issues concerning the use of blocking clauses did not use irrelevant variables, but the argument can easily be lifted to the case where $Y \neq \emptyset$.

V. Implementation

We have implemented the calculus of Section IV in our new tool DUALIZA implemented from scratch in C available at http://fmv.jku.at/dualiza. The tool counts and prints the number of models, optionally projected on a set of relevant variables. It can also act as a simple SAT solver, or enumerate all (projected) partial models, i.e., compiles a disjunctive representation of all (projected) models.

We carefully tested our tool against state-of-the-art SAT solvers and the model counter sharpSAT [20] using a regression test suite which comes with the tool but also using fuzz testing [28]. Beside the main CNF-based back-end implementing of our calculus, we also provide our own BDD engine, which obviously only scales for small formulae, but is useful to test projected model counting.

The front-end of DUALIZA reads Boolean formulae in a simple format, circuits in AIGER format [29], or CNF in DIMACS format, and encodes them into CNF using the Plaisted-Greenbaum transformation [22] after flattening the internal circuit representation. As future work we plan to further optimize the internal circuit representation. The same procedure is applied to the negated formula to obtain a dual representation.

The resulting primal and dual CNFs (P and N) are individually preprocessed by bounded variable elimination [21], which is restricted to only eliminate irrelevant variables ($S \cup T$). We will add more preprocessing and in particular more dedicated preprocessing techniques such as those from [15] in the future.

The core engine of DUALIZA uses standard data structures and algorithms implemented in state-of-the-art CDCL solvers, including watched literals, activity-based decision heuristics (actually VMTF), frequent garbage collection of learned clauses, and also frequent restarts (as long as no variable is flipped). An important optimization is to allow picking decisions in an arbitrary order until a first model is found. Then we restart without counting this first model and afterwards enforce splitting on relevant variables first. Due to phase saving this first model is found again instantly.

The source code marks the implementation of every rule of our calculus through macros ("`RULE`"), which allows to gather statistics about their application and also can be used to produce traces of the application of our calculus on concrete examples ("`dualiza -l | grep RULE`").

VI. Experiments

Consider again the example in Figure 1. The solver trace (enabled with "`-l`") shows the decision level (number of decisions plus flipped decisions) in the 3rd column, followed by the name of the rule and a running counter for each rule. The last column gives the (encoded) flipped or added literal.

The simplest interesting example is a CNF formula consisting of a single clause of n variables. In dual mode such a formula is solved by dual propagation alone. Consider for $n = 4$ the simple Boolean formula listed on the left hand side of Figure 1 in the formula input format of DUALIZA. In dual mode DUALIZA can enumerate and count the number of such a formula instantly, i.e., for $n = 10000$ the number of models $2^{10000} - 1$ is computed in 0.16 seconds. Disabling dual mode ("`--no-dual`") leads to exponential run-times in n, since our implementation requires that all variables are assigned before detecting that P is empty (rules BP1F and BP1L), as it is

common in SAT solvers. Already for $n = 30$ it takes more than 215 seconds when only applying our flipping rules and the procedure does not terminate within one hour when only using blocking clauses (even with the subsumption check described in Section VIII enabled).

For component-based model counting as implemented in the state-of-the-art exact model counter sharpSAT [20] such single clause instances are trivial too (10.58 seconds for $n = 10000$), since for each decision in one branch the formula is satisfied and in the other branch the clause is reduced in size. Actually such a solver could in principle instantly determine the number of models as soon as a component consists of a single clause.

Thus in order to show the orthogonal strength of our approach compared to component-based model counting, we also experimented with formulae where splitting on variables does not partition the formula into disconnected components. Consider for $n = 4$ the following formula

```
(x1 | x2 | x3 | x4) |
(x5 = x2 ^ x3 ^ x4) |
(x6 = x1 ^ x3 ^ x4) |
(x7 = x1 ^ x2 ^ x4) |
(x8 = x1 ^ x2 ^ x3)
```

The first row is similar to the previous example. The other equalities evaluate to true if the new variable on the left is identical to the parity ("^" denotes XOR) of different sets of three variables. The number of models is $2^{2n} - 1$, e.g., only the assignment where the first n variables are false and the second half are true is not a model. Our implementation takes 244.37 seconds to compute $2^{10000} - 1$ models for $n = 5000$, while sharpSAT shows exponential behavior and can only compute the model count up to $n = 21$ within one hour.

In a related experiment we took the 127 satisfiable CNF benchmarks from the main track of the SAT Competition 2017, for which at least one solver produced a correct witness during the competition within 5000 seconds (on a slightly faster machine). Our tool finds at least one model for 60 benchmarks within the same time limit (best solver in the competition 102), and provides an exact model count for 22 in primal mode and 16 in dual mode. This shows that dual reasoning without projection is in our current implementation not really effective for benchmarks in CNF. However, sharpSAT could only provide an exact model count for one benchmark ("9_38"), runs out of memory on 10 and actually also produced two discrepancies (claimed that both "ak128modbtbg2msisc" and "UCG-20-10p1" have no solutions).

The most recent paper on *projected* model counting [7] used random benchmarks and a set of planning benchmarks, which are only available in CNF. It turns out that our tool is not really competitive on these instances without component reasoning. Due to the non-structural CNF input format, dual propagation is not effective and our tool in essence enumerates all (projected) models explicitly. We are exploring other potential sources of benchmarks for projected model counting, including AIGER circuits from hardware model checking.

Our experiments used an Intel Xeon E5-2620 v4 CPU at 2.10GHz (turbo-mode disabled) with memory limit of 31 GB.

VII. Related Work

Due to its wide practical applicability, research on propositional model counting exhibits an impressive diversity, as demonstrated by the following list of related work.

Exact model counting. A widely used paradigm is that of splitting the formula into subformulae called *components* with disjoint variable sets which are then processed separately [18]. The authors also demonstrate the need for so-called *good learning* in contrast to *nogood learning* realized by CDCL. Combining component caching and clause learning resulted in a significant performance gain [19]. In sharpSAT [20], *implicit BCP* and an improved component caching additionally reduces search space and cache size. The caching scheme was adapted to support parallelization [30] and distributivity [31]. For a survey on exact model counting algorithms we refer to [32].

Approximate model counting. In some applications, such as probabilistic reasoning, an approximated model count would suffice. Let us mention two paradigms originating from probability theory. The first is based on sampling [33], [34], while in the second (short) XOR constraints are added to the formula until it becomes unsatisfiable [35], [36]. Exact and approximate model counting algorithms are compared in [37].

Alternative methods. All mentioned exact model counters so far are based on the DPLL algorithm [38] or an enhancement thereof. An alternative approach consists in compiling the formula into a language in which model counting is tractable [39], [40] applied in, e.g., conformant planning [41]. In theory structure-based approaches can for instance also make use of the community structure of the formula [42] or the structure of the hypergraph associated with the formula [43], [44].

VIII. Conclusion

We have presented a dual procedure for projected model counting taking into account the formula as well as its negation. We devised an efficient good learning mechanism based on the detection of partial models, which is exempt from satisfiability checks and clause watching. This method enables the pruning of a potentially large portion of the search space. Formulae with large satisfying subspaces of the search space benefit most. For these, the learned goods tend to be short. We introduced the concepts of flipping and discounting to remember the models found at the actual decision level. This allows us to jump back over branches in which models were found by simply subtracting their count from the number of models found so far. Completeness of CDCL guarantees that these models are found again at a later stage. Flipping and discounting render the use of blocking clauses superfluous and thus prevent an additional growth of the formula.

We have preliminary ideas on how to handle backjumping and redundant dual clause learning for dual conflicts. This has the potential to shrink partial models substantially, particularly in combination with discounting. In the current calculus this situation is addressed by backtracking (BN0F and BN0L).

Our method prioritizes decisions of the relevant variables over the irrelevant variables. By adopting the search strategy utilized in [13], we might be able to relax this restriction.

Note that with every relevant flipped decision the models grow larger. We plan to investigate whether overall backjumping upon a conflict in N can compensate restrictions imposed by our branching heuristics. Moreover, the algorithm presented in [13] allows to remove blocking clauses eagerly and explicitly as soon they are not needed anymore. This is not captured in our calculus yet. In our implementation we provide a poor-man's version simulating this approach partially, by trying to subsume previous blocking clauses by a new blocking clause.

Another important future extension of our calculus is to capture component reasoning, which can give exponential speed-ups orthogonal to what can be achieved by dual reasoning.

Beside the more technical challenge to improve the CNF encoding through circuit optimizations and extending and applying more preprocessing techniques to the generated CNFs individually, we also want to explore more general preprocessing techniques which jointly work on the dual representation, taking advantage of the duality property.

Finally, DUALIZA makes it easy to apply model counting in various domains. This in turn will allow the research community to obtain interesting model counting benchmarks and in general encourages more research in model counting.

REFERENCES

[1] D. Roth, "On the hardness of approximate reasoning," *Artif. Intell.*, vol. 82, no. 1-2, pp. 273–302, 1996.

[2] J. Burchard, D. Erb, and B. Becker, "Characterization of possibly detected faults by accurately computing their detection probability," in *DATE*. IEEE, 2018, pp. 385–390.

[3] F. Biondi, M. A. Enescu, A. Heuser, A. Legay, K. S. Meel, and J. Quilbeuf, "Scalable approximation of quantitative information flow in programs," in *VMCAI*, ser. LNCS, vol. 10747. Springer, 2018.

[4] S. Kölbl, G. Leander, and T. Tiessen, "Observations on the SIMON block cipher family," in *CRYPTO (1)*, ser. LNCS, vol. 9215. Springer, 2015, pp. 161–185.

[5] A. Kübler, C. Zengler, and W. Küchlin, "Model counting in product configuration," in *LoCoCo 2010*, ser. EPTCS, vol. 29, 2010, pp. 44–53.

[6] C. Zengler and W. Küchlin, "Boolean quantifier elimination for automotive configuration - A case study," in *FMICS*, ser. LNCS, vol. 8187. Springer, 2013, pp. 48–62.

[7] R. A. Aziz, G. Chu, C. J. Muise, and P. J. Stuckey, "#∃SAT: Projected model counting," in *SAT 2015*, ser. LNCS, vol. 9340. Springer, 2015, pp. 121–137.

[8] E. P. Zawadzki, A. Platzer, and G. J. Gordon, "A generalization of SAT and #SAT for robust policy evaluation," in *IJCAI*. IJCAI/AAAI, 2013, pp. 2583–2590.

[9] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Trans. Computers*, vol. 48, no. 5, pp. 506–521, 1999.

[10] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *DAC*. ACM, 2001.

[11] V. Klebanov, N. Manthey, and C. J. Muise, "SAT-based analysis and quantification of information flow in programs," in *QEST*, ser. LNCS, vol. 8054. Springer, 2013, pp. 177–192.

[12] O. Grumberg, A. Schuster, and A. Yadgar, "Memory efficient all-solutions SAT solver and its application for reachability analysis," in *FMCAD*, ser. LNCS, vol. 3312. Springer, 2004, pp. 275–289.

[13] M. Gebser, B. Kaufmann, and T. Schaub, "Solution enumeration for projected Boolean search problems," in *CPAIOR*, ser. LNCS, vol. 5547. Springer, 2009, pp. 71–86.

[14] E. Birnbaum and E. L. Lozinskii, "The good old Davis-Putnam procedure helps counting models," *J. Artif. Intell. Res. (JAIR)*, vol. 10, no. 1, pp. 457–477, 1999.

[15] J.-M. Lagniez and P. Marquis, "On preprocessing techniques and their impact on propositional model counting," *Journal of Automated Reasoning*, vol. 58, no. 4, pp. 413–481, Apr 2017.

[16] K. Fazekas, M. Seidl, and A. Biere, "A duality-aware calculus for quantified Boolean formulas," in *SYNASC 2016*. IEEE Computer Society, 2016, pp. 181–186.

[17] A. Biere, S. Hölldobler, and S. Möhle, "An abstract dual propositional model counter," in *YSIP2 2017*, ser. CEUR Workshop Proceedings, no. 1837, 2017.

[18] R. J. Bayardo and J. D. Pehoushek, "Counting models using connected components," in *AAAI-00*. AAAI Press / The MIT Press, 2000, pp. 157–162.

[19] T. Sang, P. Beame, and H. A. Kautz, "Heuristics for fast exact model counting," in *SAT*, ser. LNCS, vol. 3569. Springer, 2005, pp. 226–240.

[20] M. Thurley, "sharpSAT – counting models with advanced component caching and implicit BCP," in *SAT 2006*, ser. LNCS, vol. 4121. Springer, 2006, pp. 424–429.

[21] N. Eén and A. Biere, "Effective preprocessing in SAT through variable and clause elimination," in *SAT 2005*, ser. LNCS, F. Bacchus and T. Walsh, Eds., vol. 3569. Springer, 2005, pp. 61–75.

[22] D. A. Plaisted and S. Greenbaum, "A structure-preserving clause form translation," *J. Symb. Comput.*, vol. 2, no. 3, pp. 293–304, 1986.

[23] G. Tseitin, "On the complexity of derivation in propositional calculus," *Studies in Constructive Mathematics and Mathematical Logic*, pp. 115–125, 1968.

[24] R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T)," *J. ACM*, vol. 53, no. 6, pp. 937–977, 2006.

[25] S. Hölldobler, N. Manthey, T. Philipp, and P. Steinke, "Generic CDCL - A formalization of modern propositional satisfiability solvers," in *POS@SAT*, ser. EPiC Series in Computing, vol. 27. EasyChair, 2014.

[26] A. Goultiaeva and F. Bacchus, "Exploiting QBF duality on a circuit representation," in *AAAI*. AAAI Press, 2010.

[27] A. Goultiaeva, M. Seidl, and A. Biere, "Bridging the gap between dual propagation and CNF-based QBF solving," in *DATE*. EDA Consortium San Jose, CA, USA / ACM DL, 2013, pp. 811–814.

[28] R. Brummayer, F. Lonsing, and A. Biere, "Automated testing and debugging of SAT and QBF solvers," in *SAT 2010*, ser. LNCS, vol. 6175. Springer, 2010, pp. 44–57.

[29] A. Biere, "The AIGER And-Inverter Graph (AIG) format version 20071012," FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, Tech. Rep., 2007.

[30] J. Burchard, T. Schubert, and B. Becker, "Laissez-faire caching for parallel #SAT solving," in *SAT 2015*, ser. LNCS, vol. 9340. Springer, 2015, pp. 46–61.

[31] ——, "Distributed parallel #SAT solving," in *CLUSTER*. IEEE Computer Society, 2016, pp. 326–335.

[32] A. J. d. R. Morgado and J. Marques-Silva, "Algorithms for propositional model enumeration and counting," INESC-ID, Tech. Rep. 39, Feb 2005.

[33] S. Chakraborty, K. S. Meel, and M. Y. Vardi, "A scalable approximate model counter," in *CP*, ser. LNCS, vol. 8124. Springer, 2013.

[34] C. P. Gomes, J. Hoffmann, A. Sabharwal, and B. Selman, "From sampling to model counting," in *IJCAI 2007*, 2007, pp. 2293–2299.

[35] D. Achlioptas and P. Theodoropoulos, "Probabilistic model counting with short XORs," in *SAT*, ser. LNCS, vol. 10491. Springer, 2017.

[36] C. P. Gomes, A. Sabharwal, and B. Selman, "Model counting: A new strategy for obtaining good bounds," in *AAAI*. AAAI Press, 2006.

[37] ——, "Model counting," in *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications, 2009, vol. 185, pp. 633–654.

[38] M. Davis, G. Logemann, and D. W. Loveland, "A machine program for theorem-proving," *Commun. ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[39] P. Beame and V. Liew, "New limits for knowledge compilation and applications to exact model counting," in *UAI*. AUAI Press, 2015.

[40] A. Darwiche and P. Marquis, "A knowledge compilation map," *JAIR*, vol. 17, pp. 229–264, 2002.

[41] H. Palacios, B. Bonet, A. Darwiche, and H. Geffner, "Pruning conformant plans by counting models on compiled d-DNNF representations," in *ICAPS*. AAAI, 2005, pp. 141–150.

[42] R. Ganian and S. Szeider, "Community structure inspired algorithms for SAT and #SAT," in *SAT 2015*, ser. LNCS, vol. 9340. Springer, 2015, pp. 223–237.

[43] F. Capelli, A. Durand, and S. Mengel, "Hypergraph acyclicity and propositional model counting," in *SAT 2014*, ser. LNCS, vol. 8561. Springer, 2014, pp. 399–414.

[44] F. Capelli, "Understanding the complexity of #SAT using knowledge compilation," in *LICS*. IEEE Computer Society, 2017, pp. 1–10.