EPiC Series in Computing

Volume 65, 2019, Pages 113–126

GCAI 2019. Proceedings of the 5th Global Conference on Artificial Intelligence

Combining Conflict-Driven Clause Learning and Chronological Backtracking for Propositional Model Counting

Sibylle Möhle and Armin Biere

Johannes Kepler University Linz, Austria

Abstract

In propositional model counting, also named #SAT, the search space needs to be explored exhaustively, in contrast to SAT, where the task is to determine whether a propositional formula is satisfiable. While state-of-the-art SAT solvers are based on non-chronological backtracking, it has also been shown that backtracking chronologically does not significantly degrade solver performance. Hence investigating the combination of chronological backtracking with conflict-driven clause learning (CDCL) for #SAT seems evident. We present a calculus for #SAT combining chronological backtracking with CDCL and provide a formal proof of its correctness.

1 Introduction

The task of computing the number of models of a propositional formula, also referred to as #SAT, has various applications in hardware and software verification [5, 4, 12, 15, 11] as well as cryptography[16]. Classical applications are found in the area of probabilistic reasoning [24] as well as Bayesian networks [2, 19, 27] which are adopted in medical diagnosis and planning. A counting characterization of diagnoses is presented in [18]. Propositional model counting finds further applications in product configuration [32, 17] and planning [1, 31].

Challenges in model counting. In contrast to SAT solving, where the search terminates as soon as a satisfying variable assignment is found, in #SAT the search space needs to be explored exhaustively. State-of-the-art SAT solvers implement conflict-driven clause learning (CDCL) [22, 28], i.e., in case of a conflict they learn a clause which might be used to *backjump* or *backtrack non-chronologically* to a potentially much smaller decision level by unassigning all literals at greater decision levels. This allows them to escape search space regions without solution. In the context of #SAT, however, backjumping might cause an erroneous model count since already counted models might be found again.

This problem does not occur in the Davis-Putnam-Logeman-Loveland (DPLL) algorithm [9] where only *chronological backtracking*, i.e., backtracking to the previous decision level, is applied. However, the downside of DPLL consists of the fact that the solver is not able to escape regions of the search space without solution easily.

One therefore might ask whether chronological backtracking and CDCL might be combined in order to benefit from the strengths of both methods. A second motivation is given by the fact that CDCL may lead to redundant work since after backjumping assignments discarded might be repeated [30] which has a greater negative impact on solver performance in #SAT.

Combining CDCL and chronological backtracking. A partial answer to this question was provided in [23]. In this paper, chronological backtracking was enabled under certain conditions violating invariants considered crucial to CDCL. In CDCL with non-chronological backtracking the current variable assignment is represented by the trail which is a sequence of literals. The trail is partitioned into subsequences of literals between decisions identified by a decision level which is assigned to the literals contained in them. One of the violated invariants says that the literals on the trail are ordered in ascending order with respect to their decision level.

This is not the case in [23] anymore. Assume the formula under consideration evaluates to false under the current assignment. A clause is learned which is used to determine the jump level, i.e., the decision level to which the solver backjumps. Let ℓ be the literal of the learned clause propagated after backjumping. It is assigned to the jump level which, after backtracking chronologically to the backtrack level, might be lower than the decision level of another literal preceding ℓ on the trail. In this case variable assignments at decision levels higher than the jump level might conflict with the current assignment. This issue is dealt with by introducing a variant of backtracking in which no blocks of literals are unassigned but only the literals with decision level greater than the backtrack level, which need not be consecutive anymore.

To define more precisely the concepts introduced in [23], in [21] we provided a formalization and formal proof and generalization of the procedure. Our goal was to understand which invariants are crucial and which are redundant and to empirically confirm the effectiveness of chronological backtracking. A second motivation, which is explored in depth in this paper, was the potential use of chronological backtracking for model counting, i.e., #SAT.

In [21], we characterized five invariants, three of which are violated by chronological backtracking. Still the procedure remains correct and we proved empirically that always executing chronological backtracking in combination with CDCL does not degrade significantly solver performance confirming the suitability of chronological backtracking for model counting.

Exact propositional model counting. The state of the art in exact propositional model counting goes back to a paradigm presented in [14]. In this method the formula under consideration is split into subformulae over disjoint sets of variables which are then solved independently and their results multiplied to yield the model count of the formula. In this work the need for so-called *good learning* is identified. Several modern #SAT solvers implement this paradigm [25, 26, 29], a parallel version [6] as well as a distributed version [7] are available. Another method is based on knowledge compilation [8] in which the formula is transformed into another language in which model counting can be executed efficiently.

A totally different approach, inspired by [13, 10], was presented in [3, 20]. It is *dual*, i.e., it takes as input a formula together with its negation. The basic idea is based on the fact that a formula evaluates to $\top$ under a given variable assignment if and only if its negation evaluates to $\bot$ under the same assignment. This method also enables the detection of partial models.

Our contribution. In this work we extend our framework [21] to serve propositional model counting and provide a formal proof of its correctness. We present a counting algorithm based on model enumeration. Basically, our procedure yields a formula which is logically equivalent to the formula under consideration. The generated formula is a disjunction of pairwise contradicting conjunctions of literals, hence its model count can be determined in polynomial time.

While model enumeration and model counting are two different yet related tasks, our method is based on the first one since it facilitates reasoning about the method's properties in the fol-

lowing manner. Our argument is based on the concepts of *pending search space*, i.e., the variable assignments not yet tested, and *pending models*, i.e., the models not yet found. Obviously the two concepts are related, and the pending models are contained in the pending search space. In an implementation therefore one might decide to sum up the number of the detected models instead of recording them explicitly.

First, we extend the original calculus [21] by one rule, to capture the situation where a model is found. Second, the states now include all the (partial) models found at any point in time of the search. Compared to [20] this extension is not dual. It also exclusively uses chronological backtracking which renders blocking clauses superfluous. As a consequence, it does not need the concepts of *flipping* nor *discounting*, as introduced in [20]. In [20] we also showed by an example that combining dual reasoning and the use of blocking clauses might result in models counted twice. We also provided another example in which combining discounting and blocking clauses leads to the loss of models, i.e., a discounted model m is not detected anymore due to a blocking clause. These issues do not exist anymore in our new framework.

2 Preliminaries

Let F be a propositional formula over the set of variables V. We call *literal* a variable $v \in V$ or its negation $\neg v$. The variable of a literal ℓ is obtained by $V(\ell)$. We denote with $\bar{\ell}$ the complement of the literal ℓ, i.e., $\bar{\ell} = \neg\ell$, supposing $\neg\neg\ell = \ell$.

A *clause* is a disjunction of literals, and a *cube* is a conjunction of literals. A formula in *Conjunctive Normal Form (CNF)* is a conjunction of clauses, whereas a *Disjoint Sum-of-Products (DSOP)* formula is a disjunction of pairwise contradicting cubes. We interpret CNF formulae as sets of clauses and DSOP formulae as sets of cubes where convenient. By writing $C \in F$ we refer to a clause or cube C in a formula F. Analogously we write $\ell \in C$ if ℓ is a literal in a clause or cube C. The empty CNF formula and the empty cube are represented by $\top$, the empty DSOP formula and the empty clause are denoted by $\bot$.

A *trail*, written $I = \ell_1 \dots \ell_n$, is a sequence of literals with no duplicate variables which may also be interpreted as the conjunction of its literals. By $V(I)$ the variables of the literals on the trail I are obtained. We denote the empty trail by ε. Trails may be concatenated, $I = JK$, provided $V(J) \cap V(K) = \emptyset$. We write $J \leqslant I$ if J is a subsequence of I and $J < I$ if furthermore $J \neq I$. The position of literal ℓ on the trail I is denoted by $\tau(I, \ell)$. We may interpret I as a set of literals and write $\ell \in I$ to refer to the literal ℓ on I. Similarly to [20], the decision literals on the trail are marked by a superscript, e.g., ℓ^d denoting open "left" branches in the sense of DPLL. Flipping the value of a decision can be seen as closing the corresponding left branch and starting the "right" branch. Thus its decision literal ℓ^d becomes a *flipped literal* $\bar{\ell}$.

The *decision level function* $\delta\colon V \mapsto \mathbb{N} \cup \{\infty\}$ returns the decision level of a variable $v \in V$. We define $\delta(v) = \infty$ if v is unassigned, and δ is updated whenever a variable is assigned or unassigned. The extension to literals ℓ, clauses C and trails I is straightforward by defining $\delta(\ell) = \delta(V(\ell))$, $\delta(C) = \mathsf{max}\{\delta(\ell) \mid \ell \in C\}$ for $C \neq \bot$ and $\delta(I) = \mathsf{max}\{\delta(\ell) \mid \ell \in I\}$ for $I \neq \varepsilon$. We further define $\delta(\bot) = \delta(\varepsilon) = 0$. We write $\delta[\ell \mapsto d]$ to denote the updated function δ in which $V(\ell)$ is assigned to decision level d. Similarly, by $\delta[I \mapsto \infty]$ all literals on the trail I are unassigned. We may write $\delta \equiv \infty$ as a shortcut. The function δ is left-associative, i.e., $\delta[I \mapsto \infty][\ell \mapsto e]$ first unassigns all literals on I and then assigns literal ℓ to decision level e.

By $\delta(L) = \{\delta(\ell) \mid \ell \in L\}$ we denote the set containing the decision levels of its elements. The set of decision literals on the trail I is denoted by $\mathsf{decs}(I) = \{\ell \mid \ell^d \in I\}$ and the set containing their complements by $\overline{\mathsf{decs}(I)} = \{\bar{\ell} \mid \ell \in \mathsf{decs}(I)\}$. By $I_{\leqslant n}$ we refer to the subsequence of I containing only the literals $\ell \in I$ where $\delta(\ell) \leqslant n$. Analogously, the set $\mathsf{decs}(I)$ may be

τ	0	1	**2**	3	**4**
I	1	2	**3**	4	**5**
δ	0	1	**0**	2	**1**

τ	0	1	**2**	3
I	1	2	**3**	5
δ	0	1	**0**	1

τ	0	1
I	1	3
δ	0	0

Figure 1: In the trail I on the left, from the three trails shown, literals 3 and 5 are placed out of order. In fact, their decision level δ is lower than the decision level of a literal preceding them on the trail, i.e., with lower position τ. The trails in the middle and on the right show the results of backtracking to decision levels 1 and 0. When backtracking to the *backtrack level* b, only literals ℓ with $\delta(\ell) > b$ are removed from the trail, while the assignment order is preserved.

restricted to decision literals ℓ where $\delta(\ell) \leqslant d$ by writing $\mathsf{decs}_{\leqslant d}(I) = \mathsf{decs}(I_{\leqslant d})$. We call *slice* a subsequence of I containing all literals in I having the same decision level, e.g., $\mathsf{slice}(I, n) = I_{=n}$. Backtracking to decision level i now amounts to unassigning literals $\ell \in I$ with decision level greater than i, i.e., where $\ell \in \mathsf{slice}(I, j)$ for $j > i$.

We clarify these concepts by means of an example taken from [21]. Consider the trail I on the left hand side of Fig. 1 over variables $\{1, \ldots, 5\}$ (in DIMACS format) where τ represents the position of a literal on I and δ represents its decision level. Literals 1 and 3 were propagated at decision level zero, literal 5 was propagated at decision level one. The literals 3 and 5 are *out-of-order literals*, i.e., their decision level is smaller than the one of a literal preceding them: We have $\delta(2) = 1 > 0 = \delta(3)$, whereas $\tau(I, 2) = 1 < 2 = \tau(I, 3)$. In a similar manner, $\delta(4) = 2 > 1 = \delta(5)$, and $\tau(I, 4) = 3 < 4 = \tau(I, 5)$. Moreover, $I_{\leqslant 1} = 1\,2\,3\,5$, $\mathsf{decs}(I) = \{2, 4\}$, $\mathsf{decs}_{\leqslant 1}(I) = \{2\}$, $\mathsf{slice}(I, 1) = 2\,5$. Upon backtracking to decision level one, the literals in $\mathsf{slice}(I, 2)$ are unassigned. The resulting trail is visualized in the middle of Fig. 1. Note that since the assignment order is preserved, the trail still contains one out-of-order literal, namely 3. Backtracking to decision level zero unassigns all literals in $\mathsf{slice}(I, 2)$ and $\mathsf{slice}(I, 1)$ resulting in the trail in which all literals are placed *in order*, i.e., they are ordered in ascending order with respect to their decision level, depicted on the right hand side.

A *total assignment* is a mapping from the set of variables V to the truth values 0 and 1. The trail I may also be interpreted as a *partial assignment* where $I(\ell) = 1$ iff $\ell \in I$ and $I(\ell) = 0$ iff $\neg\ell \in I$. Thus, $I(\ell)$ is undefined if $V(\ell) \notin V(I)$. Analogously, $I(C)$ and $I(F)$ are defined. We denote with $V - I = V \setminus V(I) \subseteq V$ the set of variables in V and not in I. By $2^{|V-I|}$ we denote the number of total assignments covered by the partial assignment represented by I.

The *residual* of a formula F under the trail I is referred to by $F|_I$. It is obtained by replacing the literals $\ell \in I$ in F by $\top$ and their negation with $\bot$. Analogously, $C|_I$ for a clause C is defined. As an example, let $F = (a \vee b) \wedge (\bar{b} \vee c)$ be a formula and $I = a\,b$ the current trail. Then the residual of F under I is given by $F|_I = (c)$.

We say that the trail I *satisfies* a formula F, denoted by $I \models F$, if $I(F) \equiv \top$, i.e., $F|_I = \top$. Then I is called a *model* of F. The *model count* of F, denoted by $\#F$, is given by the number of total assignments satisfying F. If instead $I(F) \equiv \bot$, i.e., $I(C) \equiv \bot$ for a clause $C \in F$, we say that I *falsifies* F, call C the *conflicting clause* and $\delta(C)$ the *conflict level*.

A *unit clause* is a clause $\{\ell\}$ containing one single literal ℓ called *unit literal*. By $\mathsf{units}(F)$ we refer to the set of unit literals in F. Similarly, $\mathsf{units}(F|_I)$ is defined. By writing $\ell \in \mathsf{units}(F|_I)$ we denote that the unit clause $\{\ell\}$ is contained in the residual of F under I.

3 Counting via Enumeration with Chronological CDCL

Let F be a CNF formula over variables V. A DSOP representation M of F consists of cubes representing pairwise disjoint sets of (partial) models of F. Obviously, M is not unique. The model count of M and hence of F equals the sum of the model counts of the cubes in M:

$$M \text{ is a DSOP representation of } F \quad \Longrightarrow \quad \#F = \sum_{C \in M} 2^{|V-C|}$$

Let I be the current trail. We denote with $O(I)$ the *pending search space of* I, i.e., the assignments consistent with the trail not yet tested. It is given by I and its *(open) right branches* $R(I)$. Obviously $O(I)$ is not known, but it can be determined from I. Let I be of the form $I = I_0 \ell_1^d I_1 \dots \ell_m^d I_m$. Then the pending search space with respect to the trail I is given by

$$\begin{aligned} O(I) &= \bar{\ell}_1 \wedge I_{<1} \vee \bar{\ell}_2 \wedge I_{<2} \vee \bar{\ell}_3 \wedge I_{<3} \vee \dots \vee \bar{\ell}_m \wedge I_{<m} \vee I_{\leqslant m} \\ &= I_{=0} \wedge (\bar{\ell}_1 \vee I_{=1} \wedge (\bar{\ell}_2 \vee I_{=2} \wedge (\bar{\ell}_3 \vee I_{=3} \wedge (\dots \vee I_{=m-1} \wedge (\bar{\ell}_m \vee I_{=m}) \dots)))) \end{aligned}$$

which, if multiplied out, obviously is a DSOP (since $I_{=i}$ contains ℓ_i). Its cubes represent pairwise disjoint sets of total assignments. More generally, we define the pending search space of a trail I as

$$O(I) = I \vee R(I) \quad \text{where} \quad R(I) = \bigvee_{\ell \in \mathsf{decs}(I)} R_{=\delta(\ell)}(I) \quad \text{and} \quad R_{=\delta(\ell)}(I) = \bar{\ell} \wedge I_{<\delta(\ell)} \quad \text{for } \ell \in \mathsf{decs}(I)$$

denotes the right branch of the decision ℓ at decision level $\delta(\ell)$. We also define $R_{\mathsf{op}\, n}(I) = R(I_{\mathsf{op}\, n})$, where $\mathsf{op} \in \{\leqslant, \geqslant, <, >, =\}$. Then the *pending models* of F, i.e., the models of F contained in $O(I)$, are given by $F \wedge O(I)$. Assuming the models of F already found are represented by M, then $M \vee (F \wedge O(I)) \equiv F$. The pending search space $O(I)$ and the found models M are both DSOPs as well as their disjunction $M \vee O(I)$, assuming $M \wedge O(I) \equiv \bot$. The model count of F therefore is given by

$$\#F \quad = \quad \#(F \wedge O(I)) \quad + \quad \sum_{C \in M} 2^{|V-C|}$$

Example 1. Let F be a formula and let the trail be $I = abcd^d efgh^d ijk^d lmn^d op^d qrs$ with decisions $\mathsf{decs}(I) = \{d, h, k, n, p\}$ where $\delta(d) = 1$, $\delta(h) = 2$, $\delta(k) = 3$, $\delta(n) = 4$, and $\delta(p) = 5$. Let the further decision levels be $\delta(a) = \delta(b) = \delta(c) = \delta(f) = 0$, $\delta(e) = \delta(g) = \delta(m) = 1$, $\delta(i) = \delta(j) = \delta(l) = 2$, $\delta(s) = 3$, $\delta(o) = 4$, and $\delta(q) = \delta(r) = 5$. The slices are $I_{=0} = abcf$, $I_{=1} = d^d egm$, $I_{=2} = h^d ijl$, $I_{=3} = k^d s$, $I_{=4} = n^d o$, $I_{=5} = p^d qr$. We have

$$O(I) \;=\; I \vee \bigvee_{\ell \in \mathsf{decs}(I)} (\bar{\ell} \wedge I_{<\delta(\ell)}) \;=\; I \vee (\bar{d} \wedge I_{<1}) \vee (\bar{h} \wedge I_{<2}) \vee (\bar{k} \wedge I_{<3}) \vee (\bar{n} \wedge I_{<4}) \vee (\bar{p} \wedge I_{<5})$$

visualized in Fig. 2. The dashed lines below the decision literals denote the assignments where the respective decision literal is flipped, i.e., together with I they form the pending search space.

Now assume $I(F) = \top$. We assume we backtrack chronologically and flip the last decision literal p. We get $J = abcd^d efgh^d ijk^d lmn^d os\bar{p} = I_{<5}\bar{p}$ as shown at the bottom of Fig. 2 and

$$O(J) \;=\; J \vee \bigvee_{\ell \in \mathsf{decs}(J)} (\bar{\ell} \wedge I_{<\delta(\ell)}) \;=\; (\bar{d} \wedge I_{<1}) \vee (\bar{h} \wedge I_{<2}) \vee (\bar{k} \wedge I_{<3}) \vee (\bar{n} \wedge I_{<4}) \vee (\bar{p} \wedge I_{<5})$$

where $\delta(\bar{p}) = 4$. Note that $O(J) \vee I = O(I)$.

We have $O(I) = O(J) \vee I$. Thus $F \wedge O(I) = F \wedge O(J) \vee F \wedge I = F \wedge O(J) \vee I$ since $I \models F$ and the pending models of F with respect to J are exactly the pending models of F with respect to I minus the models of F represented by I.

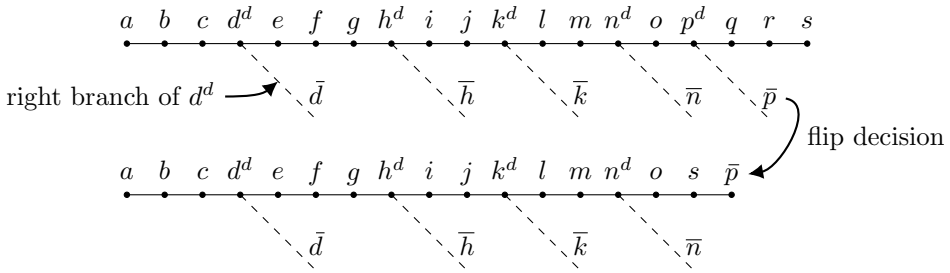

Figure 2: Trail after backtracking chronologically in Back{True, False}. The trail itself together with the dashed lines represents the pending search space, i.e., the assignments not yet tested.

4 Calculus

We describe our calculus as a state transition system where non-terminal states are denoted by (F, I, M, δ) in which F is a CNF formula over variables V, I denotes the current trail and M the DSOP containing the models found so far. Finally, δ denotes the decision level function. The initial state is given by $(F, \varepsilon, \bot, \delta_0)$ where F, ε and $\bot$ denote the original formula, the empty trail and the empty DSOP and $\delta_0 \equiv \infty$. The terminal state is equal to M which is a DSOP representation of F. The rules are listed in Fig. 3. Next we provide explanations of the rules, and in Sect. 5 we proof their correctness.

EndTrue / EndFalse. If the trail I satisfies F and there are no decision literals left on I, it is added to M and the search terminates since the search space has been traversed exhaustively. If the trail I falsifies F, there exists a clause $C \in F$ such that I falsifies C. If in addition the conflict level $\delta(C)$ is zero, the pending search space is given by $O(I) = I_{\leqslant 0}$. We have $I_{\leqslant 0}(F) = \bot$, i.e., there are no more assignments to be tested, and the search terminates.

Unit. The trail I neither satisfies nor falsifies F and there exists a clause $C \in F$ that becomes a unit clause $\{\ell\}$ under I. The trail I is extended by ℓ which is assigned to decision level $\delta(C \setminus \{\ell\})$.

BackTrue. The trail I is a model of F and is added to M. We define D as the clause consisting of all negated decision literals on I. There exist subsequences P and Q of I such that $I = PQ$ and $\delta(P) = \delta(I) - 1 = e$ and a literal $\ell \in D$ such that $D|_P = \{\ell\}$. We use D as a kind of "reason" and backtrack chronologically to decision level e. This involves unassigning all literals with decision level greater than e. For instance, $\bar{\ell} \in \mathsf{decs}(I)$ and $\delta(\bar{\ell}) = e + 1$ and upon backtracking $\bar{\ell}$ is unassigned. After backtracking we propagate ℓ and assign it decision level $\delta(D \setminus \{\ell\}) = e$ resulting in the trail $PK\ell$ where $\delta(PK\ell) = e$.

BackFalse. There exists a clause $C \in F$ such that $C|_I = \bot$. The conflict level is $\delta(C) > 0$, i.e., there is a decision left on I. We can determine (usually by way of conflict analysis[1] or just using the negated decisions) a clause D for which $F \wedge \overline{M} \models D$ such that $\delta(D) = c > 0$ and whose residual is unit, e.g., $\{\ell\}$, at jump level $j = \delta(D \setminus \{\ell\}) < c$. In fact, upon backtracking to any decision level b where $j \leqslant b < c$ the residual of D under the trail becomes unit. This holds in particular for $b = c - 1$. There exist subsequences P and Q of I such that $I = PQ$ and $\delta(P) = b$ and a literal $\ell \in D$ such that $D|_P = \{\ell\}$. We backtrack to decision level b using D as a reason. As for rule BackTrue, all literals with decision level greater than b are unassigned

[1] In contrast to [21] we require ℓ to be the negation of the last decision which corresponds to learning the decision clause instead of the more common first UIP clause. Lifting this restriction is part of future work.

EndTrue:	$(F, I, M, \delta) \rightsquigarrow_{\mathsf{EndTrue}} M \vee I$ if $F\|_I = \top$ and $\mathsf{decs}(I) = \emptyset$
EndFalse:	$(F, I, M, \delta) \rightsquigarrow_{\mathsf{EndFalse}} M$ if exists $C \in F$ and $C\|_I = \bot$ and $\delta(C) = 0$
Unit:	$(F, I, M, \delta) \rightsquigarrow_{\mathsf{Unit}} (F, I\ell, M, \delta[\ell \mapsto a])$ if $F\|_I \neq \top$ and $\bot \notin F\|_I$ and exists $C \in F$ with $\{\ell\} = C\|_I$ and $a = \delta(C \setminus \{\ell\})$
BackTrue:	$(F, I, M, \delta) \rightsquigarrow_{\mathsf{BackTrue}} (F, PK\ell, M \vee I, \delta[L \mapsto \infty][\ell \mapsto e])$ if $F\|_I = \top$ and $PQ = I$ and $D = \overline{\mathsf{decs}(I)}$ and $e + 1 = \delta(D) = \delta(I)$ and $\ell \in D$ and $e = \delta(D \setminus \{\ell\}) = \delta(P)$ and $K = Q_{\leqslant e}$ and $L = Q_{>e}$
BackFalse:	$(F, I, M, \delta) \rightsquigarrow_{\mathsf{BackFalse}} (F, PK\ell, M, \delta[L \mapsto \infty][\ell \mapsto j])$ if exists $C \in F$ and exists D with $PQ = I$ and $C\|_I = \bot$ and $c = \delta(C) = \delta(D) > 0$ such that $\ell \in D$ and $\bar{\ell} \in \mathsf{decs}(I)$ and $\ell\|_Q = \bot$ and $F \wedge \overline{M} \models D$ and $j = \delta(D \setminus \{\ell\})$ and $b = \delta(P) = c - 1$ and $K = Q_{\leqslant b}$ and $L = Q_{>b}$
Decide:	$(F, I, M, \delta) \rightsquigarrow_{\mathsf{Decide}} (F, I\ell^d, M, \delta[\ell \mapsto d])$ if $F\|_I \neq \top$ and $\bot \notin F\|_I$ and $\mathsf{units}(F\|_I) = \emptyset$ and $V(\ell) \in V$ and $\delta(\ell) = \infty$ and $d = \delta(I) + 1$

Figure 3: Rules for propositional model enumeration with chronological backtracking. The formula F takes variables in V. If I satisfies F and there is no decision at conflict level left, the search terminates (rule EndTrue). Otherwise the last decision literal is flipped (BackTrue). In case of a conflict at conflict level zero the search terminates (EndFalse). If I falsifies F at conflict level $c > 0$, a clause D is learned. The solver backtracks chronologically to decision level $c-1$ where D becomes unit and propagates its unit literal (BackFalse). If the residual of F under I is undefined, either unit propagation is applied (Unit) or a decision taken (Decide).

after which ℓ is propagated obtaining the trail $PK\ell$ where $\delta(PK\ell) = b$.

Decide. The trail I neither satisfies nor falsifies F and there are no unit literals in $F|_I$. An unassigned variable is chosen and assigned to decision level $d = \delta(I) + 1$.

Example 2. We now explain our calculus by a small example, precisely, the one for which our calculus in [20] would fail if incorrectly blocking clauses and dual reasoning were combined.

Let our formula be $F = (\overline{1} \vee 2) \wedge (1 \vee 2)$ over variables $V = \{1, 2, 3\}$ similar to DIMACS. The execution trace is shown in Fig. 4. The procedure starts with the empty trail and $M = \bot$ and by assigning all variables in V to decision level ∞ (step 0). Then, literal 3 is decided and assigned to decision level $d = 1$ (step 1). After literal 2 is decided and assigned to decision level $d = 2$, the model 32 is found (step 2). Now rule BackTrue is executed with $D = (\overline{3} \vee \overline{2})$ and $e = 1$. We backtrack chronologically by unassigning literal 2 and then propagate its complement $\overline{2}$ by assigning it to decision level 1. The found model is recorded and $M = 32$ (step 3). The unit literal $\overline{1}$ is propagated at decision level $a = 1$ due to the reason clause $(\overline{1} \vee 2)$ resulting in a conflict (step 4). The conflicting clause is $C = (1 \vee 2)$ and the conflict level is $c = 1$. Conflict analysis yields the clause $(\overline{3})$ and the jump level $j = 0$. We backtrack to decision level $b = 0$ by unassigning literals $\overline{1}$, $\overline{2}$ and 3 and propagate $\overline{3}$ by assigning it to decision level $j = 0$ (step 5),

Step	Rule	I	$\delta(1)$	$\delta(2)$	$\delta(3)$	$F\|_I$	M
0		ε	∞	∞	∞	$(\overline{1} \vee 2) \wedge (1 \vee 2)$	$\bot$
1	Decide	3^d	∞	∞	1	$(\overline{1} \vee 2) \wedge (1 \vee 2)$	$\bot$
2	Decide	$3^d 2^d$	∞	2	1	$\top$	$\bot$
3	BackTrue	$3^d\overline{2}$	∞	1	1	$(\overline{1}) \wedge (1)$	32
4	Unit	$3^d\overline{21}$	1	1	1	$\bot$	32
5	BackFalse	$\overline{3}$	∞	∞	0	$(\overline{1} \vee 2) \wedge (1 \vee 2)$	32
6	Decide	$\overline{3}1^d$	1	∞	0	(2)	32
7	Unit	$\overline{3}1^d 2$	1	1	0	$\top$	32
8	BackTrue	$\overline{31}$	0	∞	0	(2)	$32 \vee \overline{3}12$
9	Unit	$\overline{312}$	0	0	0	$\top$	$32 \vee \overline{3}12$
10	EndTrue						$32 \vee \overline{3}12 \vee \overline{312}$

Figure 4: Execution trace for Ex. 1.

after which literal 1 is decided and assigned to decision level $d = 1$ (step 6). Due to the reason clause $C = (\overline{1} \vee 2)$ with decision level $a = 1$, literal 2 is propagated by assigning it to decision level 1 and model $\overline{3}12$ is found (step 7). Rule BackTrue is executed with $D = (\overline{1})$ and $e = 0$. We backtrack chronologically by unassigning literals 2 and 1 and propagating literal $\overline{1}$ assigning it to decision level 0 (step 8). Literal 2 is propagated, the reason clause is $(1 \vee 2)$ at decision level $a = 0$, and model $\overline{312}$ is found (step 9). There are no decisions left on I, and the model found in step 9 is added to M. The search terminates with $M = 32 \vee \overline{3}12 \vee \overline{312}$ (step 10).

The choice of the decision literal is non-deterministic and might have a significant impact on the trace length as we are going to show. Assume in step 1 we decide literal 2 assigning it decision level 1. We immediately find model 2 and apply rule BackTrue with $D = (\overline{2})$ and $e = 0$, i.e., we flip the last decision literal 2 by unassigning it and assigning its complement to decision level 0. The residual of F under the trail is $(\overline{1}) \wedge (1)$, hence we apply rule Unit with $C = (\overline{1} \vee 2)$ and propagate literal $\overline{1}$ at decision level $a = 0$. We get a conflict with $C = (1 \vee 2)$ at decision level 0 and our procedure terminates with $M = 2$ after only four steps. If we start by deciding $\overline{2}$ instead, in the next step we might propagate literal $\overline{1}$ which leads to a conflict as in the last example. Rule BackFalse yields $D = (2)$ with $j = b = 0$ and after applying rule Unit and propagating literal 2 at decision level 0, we find model 2. Since the trail contains no decision literal, the procedure terminates by means of rule EndTrue in state $M = 2$ after four steps as well. Now let us decide literal 1 first. Then $I = 1^d$ and $F|_I = (2)$, literal 2 is propagated due to reason $(\overline{1} \vee 2)$ at decision level 1 and model 12 is found. By means of rule BackTrue backtracking occurs to decision level 0 and the decision literal 1 is flipped. We now have $I = \overline{1}$ and $F|_I = (2)$. Again, literal 2 is propagated resulting in $I = \overline{1}2$ which satisfies F. The trail contains no decision literals, and rule EndTrue is applied. Hence, after five steps the procedure terminates in state $M = 12 \vee \overline{1}2$. If the first decision is $\overline{1}$, analogously the following steps are executed: propagation of literal 2 finding the model $\overline{1}2$ (Unit), backtracking and flipping decision $\overline{1}$ (BackTrue), propagating literal 2 (Unit), finding model 12 and terminating with $M = \overline{1}2 \vee 12$ (EndTrue). In total, five steps are required.

We conclude this example by remarking that the trail is shortest if first literals are decided which occur either positively or negatively in all models. The worst case with respect to trail length is if literals are decided first whose variable does not appear in the formula at all. This effect is due to the nature of chronological backtracking.

(1) $\forall k, \ell \in \mathsf{decs}(I) \,.\, \tau(I,k) < \tau(I,\ell) \implies \delta(k) < \delta(\ell)$

(2) $\delta(\mathsf{decs}(I)) = \{1, \ldots, \delta(I)\}$

(3) $\forall n \in \mathbb{N} \,.\, F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$

(4) $M \vee O(I)$ is a DSOP

(5) $M \vee F \wedge O(I) \equiv F$

Figure 5: Invariants for propositional model counting with conflict-driven clause learning and chronological backtracking.

5 Proofs

For proving the correctness of our method, we make use of the invariants listed in Fig. 5 the first two of which were introduced in [21], while the third one is Inv. (3) in [21] strengthened by the negation of M. This strengthening is required to show that after backtracking upon finding a model (rule BackTrue) the propagated literals are indeed implied. Their "reason" is the clause D consisting of the negated literals of the model just found. In contrast to rule BackFalse, D is not implied by the formula F but subsumed by the negation of a cube in its DSOP representation M instead, namely the model just found.

Invariants (4) and (5) represent statements concerning the pending search space needed to show that in the end state M is logically equivalent to the formula and thus their model counts coincide. Invariant (4) states that M is DSOP, hence, its model count equals the sum of the number of models of its cubes. Invariant (5) says that all models of F are represented by the formula obtained by the disjunction of M and the pending models of F. Thus, upon termination of the procedure the second disjunct is empty, i.e., M contains all models of F.

5.1 Invariants in Non-Terminal States

Lemma 1. *Invariants* (1) – (5) *hold in non-terminal states.*

The proof is carried out by induction over the number of rule applications. Assuming that Inv. (1) – (5) hold in a non-terminal state (F, I, M, δ), we show that they are met after the transition to another non-terminal state for all rules.

Unit

Invariants (1) and (2): Except for the DSOP M contained in the state, rule Unit matches the one in [21], hence Inv. (1) and (2) are met.

Invariant (3): The argument is the same as for Inv. (3) in [21] replacing F by $F \wedge \overline{M}$ where M remains unaltered, hence Inv. (3) also holds after the application of rule Unit.

Invariant (4): We have that $M \vee O(I)$ is a DSOP and we need to show that $M \vee O(I\ell)$ is a DSOP as well. According to its definition, $O(I\ell)$ is a DSOP. We have $O(I\ell) = I\ell \vee R(I\ell) = I\ell \vee R_{\leqslant a}(I\ell) \vee R_{>a}(I\ell)$. Since $\delta(\ell) = a$ and $\ell \notin \mathsf{decs}(I\ell)$, ℓ does not occur in $R_{\leqslant a}(I\ell)$, hence $O(I\ell) = I\ell \vee R_{\leqslant a}(I) \vee R_{>a}(I) \wedge \ell$. Since $M \vee O(I)$ is a DSOP, $M \vee O(I\ell)$ is a DSOP as well.

Invariant (5): Since $F \wedge I \models \ell$, we have $F \wedge O(I) \equiv F \wedge O(I\ell)$. From this we get $M \vee F \wedge O(I) \equiv M \vee F \wedge O(I\ell) \equiv F$, and Inv. (5) holds after executing Unit.

BackTrue

Invariants (1) and (2): We need to prove that the order of the decisions left on the trail remains unaltered and no new decisions are taken. We show that (A) K contains no decision literal and (B) in the post state ℓ is not a decision literal either.

(A) We have $I = PQ$ and $K = Q_{\leqslant e}$, i.e., K is obtained from Q by removing all literals with decision level greater than e. Furthermore, for all $k \in K$, $p \in P$, we have $\tau(PK, p) < \tau(PK, k)$ and $\delta(K) \leqslant \delta(P) = e$. By the definition of decision literal and since Inv. (1) holds before applying BackTrue, the decision literal with decision level e is contained in P. Since K contains no literal with decision level greater than e, K contains no decision literal.

(B) It is sufficient to consider the case where $\delta(I) > 0$. We have $\delta(D \setminus \{\ell\}) = e = \delta(P)$ and $\delta(D) = e + 1$. According to Inv. (1) and (2) there exists exactly one decision literal for each decision level, and since $D = \overline{\mathsf{decs}(I)}$ and $\ell \in D$, $\bar{\ell} \in \mathsf{decs}(I)$. Furthermore, $\bar{\ell} \in Q$, hence $\bar{\ell}$ is unassigned upon backtracking, i.e., $\{\ell, \bar{\ell}\} \notin PK$, and $D|_{PK} = \{\ell\}$. Due to the definition of D there exists a literal $k \in D$ where $k \neq \ell$ such that $\delta(k) = \delta(\ell) = e$ for which $\tau(PK, k) < \tau(PK, \ell)$. Due to Inv. (1) and the definition of decision literal, ℓ is not a decision literal. The order of the decision literals left on the trail is not affected, hence Inv. (1) holds. Also, the remaining decisions remain unaltered, and Inv. (2) holds as well.

Invariant (3): We need to show $F \wedge \overline{(M \vee I)} \wedge \mathsf{decs}_{\leqslant n}(PK\ell) \models (PK\ell)_{\leqslant n}$ for all n. First note that the decision levels of all the literals in PK do not change while applying the rule. Only the decision level of the variable of ℓ is decremented from $e + 1$ to e. It also stops being a decision variable. Since $\delta(PK\ell) = e$, we can assume $n \leqslant e$. Observe $F \wedge \overline{(M \vee I)} \wedge \mathsf{decs}_{\leqslant n}(PK\ell) \equiv \bar{I} \wedge (F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(I))$ since ℓ is not a decision in $PK\ell$ and $I_{\leqslant e} = PK$ and thus $I_{\leqslant n} = (PK)_{\leqslant n}$ by definition. Now the induction hypothesis is applied and we get $F \wedge \overline{(M \vee I)} \wedge \mathsf{decs}_{\leqslant n}(PK\ell) \models I_{\leqslant n}$. Again using $I_{\leqslant n} = (PK)_{\leqslant n}$ this almost closes the proof except that we are left to prove $F \wedge \overline{(M \vee I)} \wedge \mathsf{decs}_{\leqslant e}(PK\ell) \models \ell$ as ℓ has decision level e in $PK\ell$ after applying the rule and thus ℓ disappears in the proof obligation for $n < e$. To see this note that $F \wedge \overline{D} \models I$ using again the induction hypothesis for $n = e + 1$. This gives $F \wedge \overline{\mathsf{decs}_{\leqslant e}(PK)} \wedge \bar{\ell} \models I$ and thus $F \wedge \overline{\mathsf{decs}_{\leqslant e}(PK)} \wedge \bar{I} \models \ell$ by conditional contraposition.

Invariant (4): We assume $M \vee O(I)$ is a DSOP and need to show that $(M \vee I) \vee O(PK\ell)$ is a DSOP as well. Now $O(I) = I \vee R_{\leqslant e+1}(I)$ since $\delta(I) = e + 1$. Further, $O(I) = I \vee R_{\leqslant e}(I) \vee R_{=e+1}(I)$. The pending search space of $PK\ell$ is equal to $O(PK\ell) = PK\ell \vee R_{\leqslant e}(PK\ell)$. But $PK = I_{\leqslant e}$ and $PK\ell = I_{\leqslant e}\ell = R_{=e+1}(I)$ since $\bar{\ell} \in \mathsf{decs}(I)$ and $\delta(\bar{\ell}) = e + 1$. In addition, $R_{\leqslant e}(PK\ell) = R_{\leqslant e}(PK)$ since $\ell \notin \mathsf{decs}(PK\ell)$ and $\delta(\ell) = e$ which gives us $R_{\leqslant e}(PK\ell) = R_{\leqslant e}(I)$. We have $O(PK\ell) = R_{=e+1}(I) \vee R_{\leqslant e}(I)$. From this we get $O(PK\ell) \vee I = O(I)$ and $(M \vee I) \vee O(PK\ell) = M \vee (I \vee O(PK\ell)) = M \vee O(I)$ which is a DSOP, and Inv. (4) holds.

Invariant (5): Given $M \vee (F \wedge O(I)) \equiv F$, we need to show that $(M \vee I) \vee (F \wedge O(PK\ell)) \equiv F$. From $O(PK\ell) \vee I = O(I)$ we get $M \vee F \wedge O(I) = M \vee F \wedge (O(PK\ell) \vee I) = M \vee (F \wedge O(PK\ell)) \vee (F \wedge I)$. But $I \models F$ and $F \wedge I \equiv I$. Therefore $M \vee F \wedge O(I) = M \vee (F \wedge O(PK\ell)) \vee I = (M \vee I) \vee F \wedge O(PK\ell) \equiv F$, and Inv. (5) is met.

BackFalse

Invariant (1) and (2): Except for the DSOP M added to the state, the fact that clause D is not added to F and that $F \wedge \overline{M} \models D$, rule BackFalse corresponds to rule Jump in [21] where $b = c - 1$. Therefore, rule BackFalse is a special case of rule Jump and Inv. (1) – (2) still hold after the execution of rule BackFalse.

Invariant (3): Let n be arbitrarily fixed. Before executing BackFalse, $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$. We need to show that $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(PK\ell) \models (PK\ell)_{\leqslant n}$. We have $I = PQ$ and $PK < I$,

i.e., $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(PK) \models (PK)_{\leqslant n}$. From $j = \delta(D \setminus \{\ell\})$, $c = \delta(D)$ and $\delta(P) = c - 1$ we get $D|_{PK} = \{\ell\}$. On the one hand $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(PK) \models \overline{D \setminus \{\ell\}}$ and on the other hand $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(PK) \models D$, therefore, by modus ponens, $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(PK) \models \ell$. Since ℓ is not a decision literal, as shown above, $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(PK) \equiv F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(PK\ell)$ and $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(PK\ell) \models (PK\ell)_{\leqslant n}$. So, Inv. (3) holds after applying rule BackFalse.

Invariant (4): Given that $M \vee O(I)$ is a DSOP, we need to show that $M \vee O(PK\ell)$ is a DSOP as well. As shown for rule BackTrue, $O(PK\ell) \vee I = O(I)$ if $\bar{\ell} \in \mathsf{decs}(I)$. Due to the premise, we have $M \wedge O(I) \equiv \bot$. Therefore, $M \wedge O(I) = M \wedge (O(PK\ell) \vee I) \equiv (M \wedge O(PK\ell)) \vee (M \wedge I) \equiv \bot$, in particular $M \wedge O(PK\ell) \equiv \bot$, and Inv. (4) holds.

Invariant (5): Given that $M \vee (F \wedge O(I)) \equiv F$, we need to show that $M \vee (F \wedge O(PK\ell)) \equiv F$ as well. Analogously to rule BackTrue we have $M \vee F \wedge O(I) = M \vee (F \wedge O(PK\ell)) \vee (F \wedge I)$. But $F \wedge I \equiv \bot$, hence $M \vee F \wedge O(I) = M \vee F \wedge O(PK\ell) \equiv F$, and Inv. (5) is met.

Decide

Invariant (1) and (2): Except for the DSOP M contained in the state, rule Decide matches exactly the one in [21], and, following the argument given there, Inv. (1) – (2) are met.

Invariant (3): Let n be arbitrarily fixed. Note that literal ℓ is a decision literal. Therefore $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(I\ell) \equiv F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(I) \wedge \ell \models I_{\leqslant n}(I) \wedge \ell \equiv (I\ell)_{\leqslant n}$, and Inv. (3) holds.

Invariant (4): Assuming $M \vee O(I)$ is a DSOP, we need to show that $M \vee O(I\ell)$ with decision literal ℓ is a DSOP as well. We have $O(I\ell) = O_{\leqslant d}(I\ell)$ where $d = \delta(I) + 1$ since $\ell \in \mathsf{decs}(I\ell)$ and $\delta(\ell) = d$. Further, $O(I\ell) = I\ell \vee R_{\leqslant d}(I\ell) = I\ell \vee R_{\leqslant d-1}(I\ell) \vee R_{=d}(I\ell) = I\ell \vee R_{\leqslant d-1}(I\ell) \vee \bar{\ell} \wedge I_{\leqslant d-1}$ where $\delta(I) = d - 1$. Observing that $\ell \notin (I\ell)_{=i}$ for $i < d$, i.e., $(I\ell)_{\leqslant i} = I_{\leqslant i}$ for $i \leqslant d - 1 = \delta(I)$, we get $O(I\ell) = I\ell \vee R(I) \vee \bar{\ell} I = I(\ell \vee \bar{\ell}) \vee R(I) = I \vee R(I) = O(I)$. This gives us $M \vee O(I\ell) = M \vee O(I)$ which due to our premise is a DSOP, and Inv. (4) holds.

Invariant (5): As just shown, $O(I\ell) \equiv O(I)$. Therefore, $M \vee F \wedge O(I\ell) \equiv M \vee F \wedge O(I) \equiv F$, and after executing rule Decide Inv. (5) still holds.

5.2 Equivalence and Model Count

Proposition 1. *If* Enumerate *reaches a terminal state M, then $M \equiv F$ and the model count of F is given by $\#F = \sum_{C \in M} 2^{|V - C|}$.*

Due to Inv. (4) and (5), in every non-terminal state M is a DSOP of models of F, but we still need to show that the resulting $I \vee M$ in EndTrue and M in EndFalse are equivalent to F.

EndTrue. Prior to applying rule EndTrue, $M \vee O(I)$ is a DSOP and $M \vee F \wedge O(I) \equiv F$. Since $\mathsf{decs}(I) = \emptyset$, we have $R(I) = \bot$ and $O(I) = I$, i.e., $M \vee I$ is a DSOP as well. Since $I \models F$, we have $F \wedge I \equiv I$ and $M \vee (F \wedge O(I)) \equiv M \vee (F \wedge I) \equiv M \vee I \equiv F$ due to the premise.

EndFalse. For showing that $M \vee F \wedge O(I) \equiv F$ we observe that $R(I) = R_{\leqslant \delta(C)}(I) = \bot$. We therefore have $O(I) = I \vee R(I) = I \vee \bot = I$. Furthermore, I falsifies F, i.e., $F \wedge I \equiv \bot$. This gives us $M \vee F \wedge O(I) \equiv M \vee F \wedge I \equiv M \equiv F$ due to the premise.

5.3 Progress

Proposition 2. Enumerate *always makes progress, i.e., in every non-terminal state a rule is applicable.*

The proof is conducted by induction over the number of rule applications. We show that in any non-terminal state (F, I, M, δ) a rule can be executed.

Assume $F|_I = \top$. If $\mathsf{decs}(I) = \emptyset$, rule EndTrue can be applied. Otherwise we choose $D = \overline{\mathsf{decs}(I)}$. We have $\delta(D) = \delta(I) = e + 1$ and due to Inv. (2) D contains exactly one decision literal ℓ such that $\delta(\ell) = e + 1$ and therefore $\delta(D) \setminus \{\ell\}) = e$. We choose P and Q such that $I = PQ$ and $e = \delta(P)$ and in particular $\ell|_Q = \bot$. After backtracking to decision level e, we have $I_{\leqslant e} = PK$ where $K = Q_{\leqslant e}$ and $D|_{PK} = \{\ell\}$. All preconditions of rule BackTrue are met.

If instead $F|_I = \bot$, there exists a clause $C \in F$ such that $C|_I = \bot$. If $\delta(C) = 0$, rule EndFalse is applicable. Otherwise, by Inv. (3) we have $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant c}(I) \equiv F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant c}(I) \wedge I_{\leqslant c} \models I_{\leqslant c}$. Since $I_{\leqslant c}(F) \equiv \bot$, also $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant c}(I) \wedge I_{\leqslant c} \equiv F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant c}(I) \equiv \bot$. We choose $\overline{D} = \mathsf{decs}(I)$ obtaining $F \wedge \overline{M} \wedge \overline{D} \wedge I_{\leqslant c} \equiv \bot$, thus $F \wedge \overline{M} \models D$. Similarly to the proof of progress given in Prop. 2 in [21], but for $b = c - 1$, it can be shown that all preconditions of rule BackFalse hold. Note that we do not explicitly add D to F since in CDCL with chronological backtracking we need no blocking clauses. We use D exclusively to determine ℓ instead.

The proof for the case where $F|_I \notin \{\top, \bot\}$ is identical to the one for Prop. 2 in [21], since apart from M in non-terminal states rules Unit and Decide are equal in both frameworks.

Since all possible cases are covered by this argument, in every non-terminal state a rule is applicable, i.e., ENUMERATE always makes progress.

5.4 Termination

Proposition 3. ENUMERATE *always terminates, i.e., no infinite state sequence is generated.*

The proof is analogous to the one in [21] with rule BackFalse replacing Jump and the additional rule BackTrue to which the observations concerning BackFalse apply as well.

6 Conclusion

The results for combining chronological backtracking with CDCL presented in [23] and its potential for propositional model counting conjectured in [21] provided the main motivation for our work. We have presented a formal calculus for propositional model counting based on these ideas and provided a formal proof of its correctness.

For our framework we chose a model enumeration approach. The main motivation therefore was the following. Let F be a formula and I the current trail. A statement of an invariant taking into account the model count alone would be something like $MC + MO = MF$ where MC denotes the number of total models of F found so far, MO denotes the number of pending total models of F and MF denotes the model count of F. But only one of the three quantities is known, namely MC, whereas the newly introduced Invariants (4) and (5) allow first for a precise characterization of both the found and pending models and second to show that the union of the two yields the models of F.

The preconditions $F|_I \neq \top$ and $\bot \notin F|_I$ may be omitted in rules Unit and Decide without compromising the procedure's correctness. The most important remaining open question is whether literal ℓ in rule BackFalse needs to be a flipped decision.

We plan to implement our rules to experimentally validate their effectiveness and to investigate possible applications in SMT and QBF. Our hope is to avoid the overhead of introducing blocking clauses in order to make use of learning. We want to extend the presented approach to projected model counting, also in combination with dual reasoning [20]. We further target component-based reasoning.

Acknowledgments Supported by Austrian Science Fund (FWF) grant S11408-N23 (RiSE) and by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

References

[1] Rehan Abdul Aziz, Geoffrey Chu, Christian J. Muise, and Peter J. Stuckey. #∃SAT: Projected model counting. In *SAT*, volume 9340 of *Lecture Notes in Computer Science*, pages 121–137. Springer, 2015.

[2] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Solving #SAT and Bayesian inference with backtracking search. *J. Artif. Intell. Res.*, 34:391–442, 2009.

[3] Armin Biere, Steffen Hölldobler, and Sibylle Möhle. An abstract dual propositional model counter. In *YSIP*, volume 1837 of *CEUR Workshop Proceedings*, pages 17–26. CEUR-WS.org, 2017.

[4] Fabrizio Biondi, Michael A. Enescu, Annelie Heuser, Axel Legay, Kuldeep S. Meel, and Jean Quilbeuf. Scalable approximation of quantitative information flow in programs. In *VMCAI*, volume 10747 of *Lecture Notes in Computer Science*, pages 71–93. Springer, 2018.

[5] Jan Burchard, Dominik Erb, and Bernd Becker. Characterization of possibly detected faults by accurately computing their detection probability. In *DATE*, pages 385–390. IEEE, 2018.

[6] Jan Burchard, Tobias Schubert, and Bernd Becker. Laissez-faire caching for parallel #SAT solving. In *SAT*, volume 9340 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2015.

[7] Jan Burchard, Tobias Schubert, and Bernd Becker. Distributed parallel #SAT solving. In *CLUSTER*, pages 326–335. IEEE Computer Society, 2016.

[8] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17:229–264, 2002.

[9] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[10] Katalin Fazekas, Martina Seidl, and Armin Biere. A duality-aware calculus for quantified Boolean formulas. In *SYNASC*, pages 181–186. IEEE, 2016.

[11] Linus Feiten, Matthias Sauer, Tobias Schubert, Alexander Czutro, Eberhard Böhl, Ilia Polian, and Bernd Becker. #SAT-based vulnerability analysis of security components - A case study. In *DFT*, pages 49–54. IEEE Computer Society, 2012.

[12] Linus Feiten, Matthias Sauer, Tobias Schubert, Victor Tomashevich, Ilia Polian, and Bernd Becker. Formal vulnerability analysis of security components. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 34(8):1358–1369, 2015.

[13] Alexandra Goultiaeva, Martina Seidl, and Armin Biere. Bridging the gap between dual propagation and CNF-based QBF solving. In *DATE*, pages 811–814. EDA Consortium San Jose, CA, USA / ACM DL, 2013.

[14] Roberto J. Bayardo Jr. and Joseph Daniel Pehoushek. Counting models using connected components. In *AAAI/IAAI*, pages 157–162. AAAI Press / The MIT Press, 2000.

[15] Vladimir Klebanov, Norbert Manthey, and Christian J. Muise. SAT-based analysis and quantification of information flow in programs. In *QEST*, volume 8054 of *Lecture Notes in Computer Science*, pages 177–192. Springer, 2013.

[16] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. Observations on the SIMON block cipher family. In *CRYPTO (1)*, volume 9215 of *Lecture Notes in Computer Science*, pages 161–185. Springer, 2015.

[17] Andreas Kübler, Christoph Zengler, and Wolfgang Küchlin. Model counting in product configuration. In *LoCoCo*, volume 29 of *EPTCS*, pages 44–53, 2010.

[18] T.K. Satish Kumar. A model counting characterization of diagnoses. In *DX 02*, pages 70–76, 2002.

[19] Wei Li, Peter van Beek, and Pascal Poupart. Performing incremental Bayesian inference by dynamic model counting. In *AAAI*, pages 1173–1179. AAAI Press, 2006.

[20] Sibylle Möhle and Armin Biere. Dualizing projected model counting. In *ICTAI*, pages 702–709. IEEE, 2018.

[21] Sibylle Möhle and Armin Biere. Backing backtracking. In *SAT*, volume 11628 of *Lecture Notes in*

Computer Science, pages 250–266. Springer, 2019.

[22] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *DAC*, pages 530–535. ACM, 2001.

[23] Alexander Nadel and Vadim Ryvchin. Chronological backtracking. In *SAT*, volume 10929 of *Lecture Notes in Computer Science*, pages 111–121. Springer, 2018.

[24] Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996.

[25] Tian Sang, Fahiem Bacchus, Paul Beame, Henry A. Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. In *SAT*, 2004.

[26] Tian Sang, Paul Beame, and Henry A. Kautz. Heuristics for fast exact model counting. In *SAT*, volume 3569 of *Lecture Notes in Computer Science*, pages 226–240. Springer, 2005.

[27] Tian Sang, Paul Beame, and Henry A. Kautz. Performing Bayesian inference by weighted model counting. In *AAAI*, pages 475–482. AAAI Press / The MIT Press, 2005.

[28] João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999.

[29] Marc Thurley. sharpSAT – counting models with advanced component caching and implicit BCP. In *SAT*, volume 4121 of *Lecture Notes in Computer Science*, pages 424–429. Springer, 2006.

[30] Peter van der Tak, Antonio Ramos, and Marijn Heule. Reusing the assignment trail in CDCL solvers. *JSAT*, 7(4):133–138, 2011.

[31] Erik Peter Zawadzki, André Platzer, and Geoffrey J. Gordon. A generalization of SAT and #SAT for robust policy evaluation. In *IJCAI*, pages 2583–2590. IJCAI/AAAI, 2013.

[32] Christoph Zengler and Wolfgang Küchlin. Boolean quantifier elimination for automotive configuration - A case study. In *FMICS*, volume 8187 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2013.