Discrete Applied Mathematics 361 (2025) 412–439

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

On enumerating short projected models☆

Sibylle Möhle [a,*], Roberto Sebastiani [b], Armin Biere [c]

[a] Research Group Automation of Logic, Max Planck Institute for Informatics, Saarland Informatics Campus E1 4, 66123 Saarbrücken, Germany
[b] Department of Information Engineering and Computer Science (DISI), University of Trento, Via Sommarive, 9, 38123 Povo (TN) Trento, Italy
[c] Faculty of Engineering, University of Freiburg, Georges Köhler Allee, 79110 Freiburg im Breisgau, Germany

ARTICLE INFO

Article history:
Received 15 September 2021
Received in revised form 21 September 2024
Accepted 26 October 2024
Available online 12 November 2024

Keywords:
Model enumeration
All-SAT
Propositional logic
Projection

ABSTRACT

Propositional model enumeration, or All-SAT, is the task to record all models of a propositional formula. It is a key task in software and hardware verification, system engineering, and predicate abstraction, to mention a few. It also provides a means to convert a CNF formula into DNF, which is relevant in circuit design. While in some applications enumerating models multiple times causes no harm, in others avoiding repetitions is crucial. We therefore present two model enumeration algorithms which adopt dual reasoning in order to shorten the found models. The first method enumerates pairwise contradicting models. Repetitions are avoided by the use of so-called blocking clauses for which we provide a dual encoding. In our second approach we relax the uniqueness constraint. We present an adaptation of the standard conflict-driven clause learning procedure to support model enumeration without blocking clauses. Our procedures are expressed by means of a calculus and proofs of correctness are provided.

1. Introduction

The *satisfiability problem of propositional logic (SAT)* consists in determining whether for a propositional formula there exists an assignment to its variables which evaluates the formula to true and which we call *satisfying assignment* or *model*. For proving that a formula is satisfiable, it is sufficient to provide one single model. However, sometimes determining satisfiability is not sufficient but all models are required. *Propositional model enumeration (All-SAT)*[1] is the task of enumerating (all) satisfying assignments of a propositional formula. It is a key task in, e. g., bounded and unbounded model checking [5,26,32,33,58,59], image computation [21,22,29,57], system engineering [62], predicate abstraction [28], and lazy Satisfiability Modulo Theories [55].

Model enumeration also provides a means to convert a formula in Conjunctive Normal Form (CNF) into a logically equivalent formula in Disjunctive Normal Form (DNF) composed of the models of the CNF formula. This conversion is used in, e. g., circuit design [35] and has also been studied from a computational complexity point of view [34,66], and in the worst case it is exponential in the size of the original formula due to its exponential blowup. If the models found are

☆ The first author's contribution was mostly carried out at the LIT Correct and Secure Systems Lab and the Institute for Formal Models and Verification, Johannes Kepler University Linz, Linz, Austria.

* Corresponding author.
E-mail address: smoehle@acm.org (S. Möhle).

[1] For the sake of readability, we use the term *All-SAT* also if not all models are required since in principle such an algorithm could always be extended to determine all models.

https://doi.org/10.1016/j.dam.2024.10.021
0166-218X/

pairwise contradicting, the resulting DNF is a Disjoint Sum-of-Product (DSOP) formula, which is relevant in circuit design [4,37], and whose models can be enumerated in polynomial time in the number of its disjuncts [40] by simply returning them, as they represent implicants of the formula. If the models found are not pairwise contradicting, the resulting formula is still a DNF but does not support polytime model counting. Our model enumeration algorithm basically executes a CNF-to-DNF conversion and, from this point of view, it can be interpreted as a knowledge compilation algorithm.

The aim of knowledge compilation is to transform a formula into another language[2] on which certain operations can be executed in polynomial time [10,12]. This can be done, for instance, by recording the trace of an exhaustive search [24,27,46], and the target language in these approaches is the deterministic Decomposable Normal Form (d-DNNF),[3] which was applied, for instance, in planning [50]. In contrast, in our work we record the models of the input formula, and the resulting formula is in d-DNNF only if the detected models are pairwise contradicting.

Enumerating models requires to process the search space exhaustively and is therefore a harder task than determining satisfiability. However, since state-of-the-art SAT solvers are successfully applied in industrial applications, it seems natural to use them as a basis for model enumeration. Modern SAT solvers implement *conflict-driven clause learning (CDCL)* [31,45,60] with *non-chronological backtracking*.[4] If a CDCL-based SAT solver is extended to support model enumeration, adequate measures need be taken to avoid enumerating models multiple times as demonstrated by the following small example.

Example 1 (*Multiple Model Enumeration*)**.** Consider the propositional formula

$$F = \underbrace{(a \vee c)}_{C_1} \wedge \underbrace{(a \vee \neg c)}_{C_2} \wedge \underbrace{(b \vee d)}_{C_3} \wedge \underbrace{(b \vee \neg d)}_{C_4}$$

which is defined over the set of variables $V = \{a, b, c, d\}$. Its total models[5] are given by $\mathsf{models}(F) = \{a\,b\,c\,d, a\,b\,c\,\neg d, a\,b\,\neg c\,d, a\,b\,\neg c\,\neg d\}$. These models may be represented by $a\,b$, i. e., they are given by all total extensions of $a\,b$.

Let our model enumerator be based on CDCL with non-chronological backtracking. Assume we first decide a, i. e., assign a the value true, and then b. This (partial) assignment $a\,b$ is a model of F. As in our previous work on propositional model counting [40], we flip the second decision literal, i. e., assign b the value false, in order to explore the second branch, upon which the literal d is forced to true in order to satisfy clause C_3. The resulting assignment $a\,\neg b\,d$ now falsifies clause C_4, i. e., sets all its literal to false. Conflict analysis yields the unit clause $C_5 = (b)$, which is added to F. The enumerator then backtracks to decision level zero, i. e., unassigns d, b and a, and propagates b with reason C_5. No literal is enforced by the assignment b, and a decision need be taken. If we choose a, F is satisfied. The model found is $b\,a$, which is the one we had found earlier.

Multiple model enumeration in Example 1 is caused by the fact that after conflict analysis the same satisfying assignment is repeated, albeit in reverse order. More generally, the same satisfying assignment might be found again if the enumerator backtracks past a flipped decision literal. Avoiding enumerating models multiple times is crucial in, e. g., weighted model counting (WMC) [11,15,17,54] and Bayesian inference [2], which require enumerating the models in order to compute their weight or probability. Another example is weighted model integration (WMI) [42,43] which generalizes WMC for hybrid domains. In some applications, repeating models might lead to inefficiency and harm scalability [62]. In the context of model counting but also relevant in model enumeration, Bayardo and Pehoushek [3] identified the need for good learning similarly to its learning counterpart in CDCL, and various measures have therefore been proposed to avoid the multiple enumeration of models.

One possibility is to rule out a model which was already found by adding a *blocking clause* to the formula [32,38,44] which in essence is the negation of the model or the decision literals in the model to be blocked [44]. Whenever a satisfying assignment is repeated, the clause blocking it is falsified, and thus this model is not enumerated again. As soon as all models are found and the relevant blocking clauses added, the formula becomes unsatisfiable. However, there might be an exponential number of models and adding a blocking clause for each of them might result in a significant negative impact on the enumerator performance. In these cases, multiple model enumeration need be prevented by other measures. Toda and Soh [63] address this issue by adopting a variant of conflict analysis which is inspired by Gebser et al. [20] and is exempt from blocking clauses.

The use of blocking clauses can also be avoided by adopting the Davis–Putnam–Logemann–Loveland (DPLL) algorithm [13]. In DPLL, after a conflict or a model the last decision literal is flipped causing the solver to find only pairwise contradicting models. This idea was applied in the context of model counting by Birnbaum and Lozinskii [7] but can readily be adapted to support model enumeration. Chronological backtracking in Grumberg et al. [21] and in part in

[2] A language in this context refers to one of the various forms a formula can be expressed in, e. g., CNF and DNF denote the languages we are mostly interested in this article.

[3] A formula is in d-DNNF, if (1) the sets of variables of the conjuncts of each conjunction are pairwise disjoint, and (2) the disjuncts of each disjunction are pairwise contradicting [12]. Whereas in its original definition a d-DNNF formula is defined as a directed acyclic graph (DAG), in this work we refer to its representation made of conjunctions, disjunctions, negations, variables, and truth values.

[4] Also referred to as *backjumping* in the literature.

[5] In total models all variables occur.

Gebser et al. [20] ensures that the search space is traversed in a systematic manner similarly to DPLL, and that the use of blocking clauses can be avoided. An apparent drawback of DPLL-based solvers, however, is that they might spend a significant amount of time in regions of the search space having no satisfying assignments, since — unlike CDCL-based solvers — they lack the possibility to escape those regions early.

This last issue can be addressed by the use of chronological CDCL introduced by Nadel and Ryvchin [39,47]. Chronological CDCL combines the power of conflict-driven clause learning with chronological backtracking. Specifically, after finding a model, the last open (left) decision literal is flipped in order to process neighboring regions of the search space, while in case of a conflict the solver is able to escape solution-less regions early. In our earlier work [40], we developed a calculus for propositional model counting based on chronological CDCL and provided a proof of its correctness. We took a model enumeration approach making our method readily applicable in the context of model enumeration without repetition. However, while finding short models is crucial in, e. g., weighted model integration [42,43], only total models are detected as is usual in CDCL-based SAT solvers. The reason for this is a simple one.

To detect when a partial assignment[6] is a model of the input formula, the SAT solver would have to carry out satisfiability checks before every decision, as done by Birnbaum and Lozinskii [7]. These satisfiability checks are expensive, and assigning the remaining variables instead is more efficient, computationally. If all variables are assigned and no conflict has occurred, the SAT solver knows to have found a model. This makes sense in SAT solving. Model enumeration, however, is a harder task, and therefore more expensive methods might pay off.

One such method is *dual reasoning* [6,38]. Our dual model counter Dualiza[7] takes as input the formula under consideration together with its negation. The basic idea is to execute CDCL on both formulae simultaneously maintaining one single trail. Whenever a conflict in the negated formula occurs, the current (partial) assignment is a model of the formula. Although developed for model counting, its adaptation for model enumeration is straightforward.

Another idea enabling the detection of short models was to check whether all total extensions of the current (partial) assignment evaluate the input formula to true before taking a decision, i. e., whether the current assignment logically entails the input formula [41,56].

Partial assignments evaluating the input formula to true represent sets of total models of the input formula. However, these sets might not be disjoint as is demonstrated by the following example.

Example 2 (*Short Redundant Models*)**.** Let $F = (a \wedge b) \vee (a \wedge c)$ be a propositional formula defined over variables $V = \{a, b, c\}$. Notice that F is not in CNF and significantly differs from the one in our previous example. Its total models are $\mathsf{models}(F) = \{a\,b\,c, a\,b\,\neg c, a\,\neg b\,c\}$. These models may also be represented by the two partial models $a\,b$ and $a\,c$. The former represents $a\,b\,c$ and $a\,b\,\neg c$, whereas the latter represents $a\,b\,c$ and $a\,\neg b\,c$. Notice that $a\,b\,c$ occurs twice.

Partial assignments evaluating the input formula to true result in blocking clauses which are shorter than the ones blocking one single total model. Adding short blocking clauses has a twofold effect. First, a larger portion of the search space is ruled out. Second, fewer blocking clauses need be added which mitigates their negative impact on solver performance. Also, short blocking clauses generally propagate more eagerly than long ones. The need for shrinking or minimizing models has been pointed out by Bayardo and Pehoushek [3] and addressed further [1,26,52]. Notice that with blocking clauses CDCL can be used as in SAT solving, while in the absence of blocking clauses it need be adapted.

The reason is as follows. If a CDCL-based SAT solver encounters a conflict, it analyzes it and *learns* a clause[8] in order to prevent the solver from repeating the same assignment which caused the conflict. This clause is determined by traversing the trail in reverse assignment order and resolving the reasons of the literals on the trail, starting with the conflicting clause, until the resolvent contains one single literal at the maximum decision level. If a model is found, the last decision literal is flipped in order to explore another branch of the search space. This leads to issues if this literal is encountered in later conflict analysis and no blocking clause was added, since in this case it is neither a decision literal nor a propagated literal.

To address this issue, Grumberg et al. [21] introduce sub-levels for flipped decision literals treating them similarly to decision literals in future conflict analysis. Similarly to Gebser et al. [20], Toda and Soh [63] limit the level to which the solver is allowed to backtrack. These measures also ensure that enumerating overlapping partial models is avoided. However, in applications where repetitions cause no harm, the power of finding even shorter models representing larger, albeit not disjoint, sets of models, can be exploited. Shorter models are also obtained in the case of model enumeration under *projection*.

If not all variables are relevant in an application, we project the models of the input formula onto the relevant variables, or, otherwise stated, we existentially quantify the irrelevant variables. Projection occurs in, e. g., model checking [58,59], image computation [21,22], quantifier elimination [9,67], and predicate abstraction [28]. The breadth of these applications highlights the relevance of projection in practice.

6 In a partial assignment not all variables occur.

7 https://github.com/arminbiere/dualiza

8 We say that a clause is learned if it is added to the formula.

Our contributions. In this article we address the task of enumerating short projected models with and without repetition. We start by presenting a CDCL-based algorithm for the case where only pairwise contradicting, i. e., *irredundant*, models are sought. Multiple model enumeration is prevented by the addition of blocking clauses to the input formula, and dual reasoning is adopted for shrinking total models. To ensure correctness of the latter, we introduce the concept of *dual blocking clauses* which provides a solution to an issue identified in our earlier work [38]. Dual reasoning in model shrinking enables us to obtain short models, and CDCL lets us exploit the strengths of state-of-the-art SAT solvers. Short models result in short blocking clauses with the potential to reduce their number and to rule out a larger portion of the search space. We express our algorithm by means of a formal calculus and provide a correctness proof. A generalization of our algorithm to the case where partial satisfying assignments are found and shrunken is presented. This generalization makes sense as we do not guarantee that our model shrinking method gives us the minimal model. We discuss the appropriate changes to our algorithm, calculus, proof, and its generalization.

We then introduce a relaxed version of our algorithm for enumerating non-contradicting, i. e., *redundant*, models. This method is exempt of blocking clauses, and consequently decision literals which were flipped after a model lack a reason. To fix this issue, we introduce an adaptation of CDCL for SAT to All-SAT. We discuss the changes to our previous algorithm needed in order to support redundant model enumeration.

This article builds on our work presented at the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT) 2020 [41]. It also uses concepts introduced by Sebastiani [56] as well as presented at the Second Young Scientist's International Workshop on Trends in Information Processing (YSIP2) 2017 [6] and the 30th International Conference on Tools with Artificial Intelligence (ICTAI) 2018 [38], the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT) 2019 [39], and the 5th Global Conference on Artificial Intelligence (GCAI) 2019 [40].

Structure of the paper. In Section 2, we introduce our notation and basic concepts. Dual reasoning is applied for shrinking models in Section 3, and an according dual encoding of blocking clauses is introduced in Section 4. After presenting our algorithm for projected model enumeration without repetition in Section 5 and providing a formalization and correctness proof and a generalization to the detection of partial models in Section 6, we turn our attention to projected model enumeration with repetition. We adapt CDCL for SAT to support conflict analysis in the context of model enumeration without the use of blocking clauses in Section 7 and discuss the changes to our method needed to support multiple model enumeration in Section 8, before we conclude in Section 10.

2. Preliminaries

In this section we provide the concepts and notation on which our presentation relies: propositional satisfiability (SAT) and incremental SAT solving, projection, and the dual representation of a formula, which constitutes the basis for dual reasoning.

2.1. Propositional satisfiability (SAT)

The set containing the Boolean constants 0 (false) and 1 (true) is denoted with $\mathbb{B} = \{0, 1\}$. Let V be a set of propositional (or Boolean) variables. A *literal* is either a variable $v \in V$ or its negation $\neg v$. We write $\overline{\ell}$ to denote the *complement* of ℓ assuming $\overline{\ell} = \neg\ell$ and $\overline{\neg\ell} = \ell$. The variable of a literal ℓ is obtained by $V(\ell)$. This notation is extended to formulae, clauses, cubes, and sets of literals.

Most SAT solvers work on formulae in *Conjunctive Normal Form (CNF)* which are conjunctions of *clauses*, which are disjunctions of literals. These SAT solvers implement efficient algorithms tailored for CNFs, such as unit propagation, which will be presented below. In contrast, a formula in *Disjunctive Normal Form (DNF)* is a disjunction of *cubes* which are conjunctions of literals. We interpret formulae as sets of clauses and write $C \in F$ to refer to a clause C occurring in the formula F. Accordingly, we interpret clauses and cubes as sets of literals. The empty CNF formula and the empty cube are denoted by 1, while the empty DNF formula and the empty clause are represented by 0.

A *total assignment* $\sigma : V \mapsto \mathbb{B}$ maps V to the truth values 0 and 1. It can be applied to a formula F over a set of variables V to obtain the *value of* F *under* σ, denoted by $\sigma(F) \in \mathbb{B}$, also written $F|_\sigma$. A sequence $I = \ell_1, \ldots, \ell_n$ with mutually exclusive variables ($V(\ell_i) \neq V(\ell_j)$ for $i \neq j$) is called a *trail*. If their variable sets are disjoint, trails and literals may be concatenated, denoted $I = I'\, I''$ and $I = I'\, \ell\, I''$. Consider as an example a set of variables $V = \{a, b, c, d, e, f, g, h\}$ and let $I' = a\, \neg b\, \neg c$, $I'' = e\, f\, \neg g\, h$, and $\ell = \neg d$ be two trails and a literal, respectively. Now $V(I') = \{a, b, c\}$, $V(I'') = \{e, f, g, h\}$, and $V(\ell) = d$. Since $V(I') \cap V(I'') = \emptyset$ and $V(\ell) \notin V(I') \cup V(I'')$, they can be concatenated obtaining, e. g., $I = I'\, I'' = a\, \neg b\, \neg c\, e\, f\, \neg g\, h$ and $I = I'\, \ell\, I'' = a\, \neg b\, \neg c\, \neg d\, e\, f\, \neg g\, h$. We treat trails as conjunctions or sets of literals and write $\ell \in I$ if ℓ is contained in I. Trails can also be interpreted as partial assignments with $I(\ell) = 1$ iff $\ell \in I$. Similarly, $I(\ell) = 0$ iff $\neg\ell \in I$, and $I(\ell)$ is undefined iff $V(\ell) \notin V(I)$. The unassigned variables in V are denoted by $V - I$ and the empty trail by ε.

We call *residual* of F under I, denoted $F|_I$, the formula $I(F)$ obtained by assigning the variables in F their truth value. If F is in CNF, this amounts to removing from F all clauses containing a literal $\ell \in I$ and removing from the remaining clauses all occurrences of $\neg\ell$. For instance, given a formula $F = (a \vee b) \wedge (\neg a \vee b \vee c)$ with $V(F) = \{a, b, c\}$ and $I = a$, $F|_I = (b \vee c)$. If $F|_I = 1$, we say that I *satisfies* F or that I is a *model* of F. If all variables are assigned, we call I a *total model* of F. Following the distinction highlighted by Sebastiani [56], if I is a partial assignment, we say that I *evaluates*

Input: formula F
Output: SAT if F is satisfiable, UNSAT if F is unsatisfiable

```
DPLL ( F )
1  I := ε
2  forever do
3    C := PropagateUnits ( F, I )
4    if C ≠ 0 then
5      if decs(I) = ∅ then
6        return UNSAT
7      else
8        flip most recent decision ℓ^d ∈ I
9    else
10     if there are unassigned variables in V(F) then
11       Decide ( F, I )
12     else
13       return SAT

PropagateUnits ( F, I )
1  while C|_I = (ℓ) for some C ∈ F do
2    I := I ℓ
3    for all clauses D ∈ F such that ¬ℓ ∈ D do
4      if D|_I = 0 then return D
5  return 0

Decide ( F, I )
1  I := I ℓ^d where V(ℓ) ∈ V(F) \ V(I)
```

Fig. 1. Davis–Putnam–Logemann–Loveland (DPLL) Algorithm. The main loop starts with exhaustive unit propagation. If a conflict occurs, the most recent decision is flipped. If there are no decisions left on the trail, the execution terminates returning UNSAT. If no conflict occurs and there are still unassigned variables, a decision is taken. Otherwise, the execution terminates returning SAT.

F to 1, written $I \vdash F$, if $F|_I = 1$, and that I *logically entails* F, written $I \models F$, or that I is a *partial model* of F, if all total assignments extending I satisfy F. Notice that $I \vdash F$ implies that $I \models F$ but not vice versa: e.g., if $F \stackrel{\text{def}}{=} (a \wedge b) \vee (a \wedge \neg b)$ and $I \stackrel{\text{def}}{=} a$, then $I \models F$ but $I \not\vdash F$, because $F|_I = (b \vee \neg b) \neq 1$. If F is in CNF without valid clauses, i.e., without clauses containing contradicting literals, then $I \vdash F$ iff $F|_I = 1$. We say that I *evaluates* F *to* 0 or that I is a *counter-model* of F, iff $F|_I = 0$. If F is in CNF, its residual under I contains the empty clause, $0 \in F|_I$.

2.2. The Davis–Putnam–Logemann–Loveland (DPLL) algorithm

The satisfiability of a propositional formula F over a set of variables V can be determined by the Davis–Putnam–Logemann–Loveland (DPLL) algorithm [13,14] depicted in Fig. 1.[9] Its main ingredient is the trail I which is iteratively extended by a literal ℓ which is either *propagated* or *decided*. In the former case, there exists a clause $C \in F$ containing ℓ in which all literals except ℓ evaluate to false under the current (partial) assignment I. The literal ℓ is called *unit literal* or *unit* and C a *unit clause*. In order to satisfy C, and thus F, the literal ℓ need be assigned the value true. After being propagated, the literal ℓ becomes a *propagation literal*, and C is called its *reason*. If after the propagation of ℓ a clause $D \in F$ becomes false, this clause is returned to indicate that a *conflict* occurred. The corresponding rule is the *unit propagation* rule (function PropagateUnits). Notice that $F|_I$ may contain multiple reasons for a unit literal, and by speaking of "its" reason we refer to the one chosen in the current execution. If a literal is decided, its value is chosen according to some heuristic by a *decision*, and it is called *decision literal*. We annotate decision literals on the trail by a superscript, e.g., ℓ^d, denoting open "left" branches (Decide). If a decision literal ℓ^d is flipped, its complement $\overline{\ell}$ opens a "right" branch. The set consisting of all decision literals on the trail I is obtained by $\text{decs}(I) = \{\ell \mid \ell^d \in I\}$.

[9] As it is common practice in the SAT community, we do not consider the pure-literal rule from the original DPLL procedure because it is considered ineffective.

In the main loop of the function DPLL, first exhaustive unit propagation is carried out (line 3). If it does not return the empty clause, a conflict has occurred. If there is no decision literal left on the trail, both the left and right branch of all decisions have been explored. The trail I cannot be extended to a satisfying assignment of F and the execution stops returning UNSAT (lines 4–6). Otherwise, the literals assigned after the most recent decision ℓ^d are removed from I and ℓ^d is flipped (line 8). If exhaustive unit propagation returns the empty clause, no conflict has occurred and there is no unit literal in $F|_I$. If not all variables are assigned a value, a decision need be taken (lines 10–11). Otherwise, the execution terminates returning SAT (line 13).

2.3. Conflict-Driven Clause Learning (CDCL)

The DPLL algorithm lacks the possibility to escape early from solution-less regions of the search space. Conflict-driven clause learning (CDCL) enables the solver to learn a clause representing the reason for the current conflict and to accordingly undo multiple decisions in one step. The trail is now partitioned into blocks, called *decision levels*, which extend from a decision literal to the last literal preceding the next decision.

The *decision level function* $\delta : V \mapsto \mathbb{N}$ assigns and alters decision levels. The decision level of a variable $v \in V$ is obtained by $\delta(v)$. If v is unassigned, we have $\delta(v) = \infty$. We extend δ accordingly to determine the decision level of literals ℓ, non-empty clauses C, and non-empty trails I, by defining $\delta(\ell) = \delta(V(\ell))$, $\delta(C) = \max\{\delta(\ell) \mid \ell \in C\}$, and $\delta(I) = \max\{\delta(\ell) \mid \ell \in I\} = \#\{\ell \mid \ell^d \in I\}$. The decision level of the trail I therefore corresponds to the number of decision literals on I, and if I contains only propagated literals, then $\delta(I) = 0$. The subsequence of I consisting of all literals with decision level smaller or equal to n, is denoted by $I_{\leqslant n}$. Accordingly, we define $\delta(L) = \max\{\delta(\ell) \mid \ell \in L\}$ for a non-empty set of literals L. If v is unassigned, we have $\delta(v) = \infty$, and $\delta(0) = \delta(\varepsilon) = \delta(\emptyset)$ for the empty clause, the empty sequence and the empty set of literals. Whenever a variable is assigned or unassigned, the decision level function δ is updated. If $V(\ell)$ is assigned at decision level d, we write $\delta[\,\ell \mapsto d\,]$. If all variables in the set of variables V are assigned decision level ∞, we write $\delta[\,V \mapsto \infty\,]$ or $\delta \equiv \infty$ as a shortcut. Similarly, if all literals occurring on the trail I are unassigned, i. e., removed from I, their decision level is assigned ∞, and we write $\delta[\,I \mapsto \infty\,] = \delta[\,V(I) \mapsto \infty\,]$. The function δ is left-associative, i. e., $\delta[\,I \mapsto \infty\,][\,\ell \mapsto d\,]$ first unassigns all variables on I and then assigns literal ℓ at decision level d.

Literals occurring before the first decision are assigned exclusively by unit propagation at decision level zero. A propagated literal is annotated with its reason, as in ℓ^C, and assigned at the current decision level $\delta(I)$. The trail can be represented graphically by the *implication graph* which is defined as follows. Decision literals are represented as nodes on the left and annotated with their decision level. Propagated literals are internal nodes with one incoming arc originating from each node representing a literal in their reason. A conflict is represented by the special node κ whose incoming arcs are annotated with the conflicting clause.

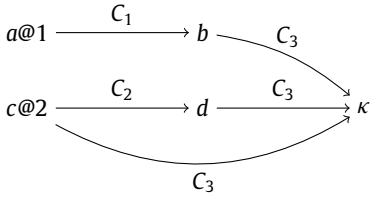
Example 3 (*Trail and Implication Graph*)**.** Consider the formula

$$F = \underbrace{(\neg a \vee b)}_{C_1} \wedge \underbrace{(\neg c \vee d)}_{C_2} \wedge \underbrace{(\neg b \vee \neg c \vee \neg d)}_{C_3}$$

over the set of variables $V = \{a, b, c, d\}$. Assume we first decide a, then propagate b with reason C_1 followed by deciding c and propagating d with reason C_2. Under this assignment, the clause C_3 is falsified. The current trail is given by

$$I = a^d\, b^{C_1}\, c^d\, d^{C_2}$$

where $\delta(a) = \delta(b) = 1$, $\delta(c) = \delta(d) = 2$, and $\mathsf{decs}(I) = \{a, b\}$. The corresponding implication graph is



As in DPLL, the CDCL algorithm, see Fig. 2, executes a main loop until either a satisfying assignment has been found or all possible assignments have been checked without finding one. It starts with exhaustive unit propagation (line 4 and function PropagateUnits). If a conflict occurs, we call the clause whose literals are set to false under I *conflicting clause*. If the trail I contains no decision literal, i. e., its decision level is zero, the execution terminates returning UNSAT (lines 6–7). Otherwise, *conflict analysis* is executed and backtracking occurs (line 9). If no conflict occurs and there are unassigned variables, a decision is taken (line 12), where the new decision literal is assigned to the decision level of I. Otherwise, the execution terminates returning SAT (line 14).

Conflict analysis is described by procedure AnalyzeConflict. Suppose the current trail I falsifies the clause $C \in F$. The basic idea is to compute a clause, let us say D, containing the negated assignments responsible for the conflict. By adding D to F, this assignment is blocked. Moreover, backtracking to the second highest decision level in D results in D

Input: formula F
Output: SAT if F is satisfiable, UNSAT if F is unsatisfiable

```
CDCL ( F )
1  I := ε
2  δ[ V ↦ ∞ ]
3  forever do
4    C := PropagateUnits ( F, I, δ )
5    if C ≠ 0 then
6      if δ(I) = 0 then
7        return UNSAT
8      else
9        AnalyzeConflict ( F, I, C, δ )
10   else
11     if there are unassigned variables in V(F) then
12       Decide ( F, I, δ )
13     else
14       return SAT

PropagateUnits ( F, I, δ )
1  while some C ∈ F is unit (ℓ) under I do
2    I := I ℓ^C
3    δ(ℓ) := δ(I)
4    for all clauses D ∈ F containing ¬ℓ do
5      if D|_I = 0 then return D
6  return 0

AnalyzeConflict ( F, I, C, δ )
1  D := Learn ( I, C )
2  F := F ∧ D
3  ℓ := literal in D at decision level δ(I)
4  j := δ(D \ {ℓ})
5  for all literals k ∈ I with decision level > j do
6    assign k decision level ∞
7    remove k from I
8  I := I ℓ^D
9  δ(ℓ) := j

Decide ( F, I, δ )
1  I := I ℓ^d where V(ℓ) ∈ V(F) \ V(I)
2  δ(ℓ) := δ(I)
```

Fig. 2. CDCL-based satisfiability algorithm. In the main loop, first exhaustive unit propagation is executed. If a conflict at decision level zero occurs, the execution terminates returning UNSAT. Otherwise, the conflict is analyzed, a clause blocking the assignment responsible for the conflict is learned and backjumping occurs. If no conflict occurred and all variables are assigned, the execution terminates returning SAT, otherwise a decision is taken.

becoming unit, and its literal with highest decision level is propagated. A main ingredient of the clause learning algorithm is *resolution* [53]. Given two clauses $(A \vee \ell)$ and $(B \vee \neg\ell)$, where A and B are disjunctions of literals and ℓ is a literal, their *resolvent* $(A \vee \ell) \otimes_\ell (B \vee \neg\ell) = (A \vee B)$ is obtained by resolving them on ℓ. The clause D is determined by a sequence of resolution steps which can be read off either the implication graph or the trail. First, the conflicting clause is resolved with the reason of one of its literals. This procedure is repeated with the reason of one literal in the resolvent and continued

until the resolvent contains one single literal at *conflict level*, which is the decision level of the conflicting clause. In the implication graph, we start with the conflict node κ and follow the edges in reverse direction to determine the next literal to resolve on. Considering the trail, we start with the conflicting clause and choose the most recent propagated literal for resolution.

Example 4 (*Trail-Based Conflict-Driven Clause Learning*)**.** Consider the situation in Example 3. The conflicting clause is $C_3 = (\neg b \vee \neg c \vee \neg d)$ with $\delta(C_3) = 2$ and $I = a^d\, b^{C_1}\, c^d\, d^{C_2}$. The most recent unit literal is d with reason C_2. We resolve C_3 with C_2 on d and obtain $C_3 \otimes_d C_2 = (\neg b \vee \neg c \vee \neg d) \otimes_d (\neg c \vee d) = (\neg b \vee \neg c)$.[10] Now $\delta(\neg b) = 1 \neq 2 = \delta(d) = \delta(C_3)$, and the clause $(\neg b \vee \neg c)$ is learned.

Following the trail in reverse assignment order gives us a deterministic sequence of resolution steps. In contrast, when determining the clause to be learned based on the implication graph, this choice need not be deterministic.

Example 5 (*Implication-Graph-Based Conflict-Driven Clause Learning*)**.** Consider the implication graph given in Example 3 for clause learning. The choice of the clauses to resolve need not be deterministic. For instance, we can resolve C_3 either on d with C_2 or on b with C_1. If we choose C_1, the resolvent $(\neg a \vee \neg c \vee \neg d)$ contains two literals at conflict level, namely $\neg c$ and $\neg d$, and a second resolution step is needed, whereas by resolving C_3 with C_2 on d first, see Example 4, one resolution step is saved.

2.4. Incremental SAT solving

The basic idea of incremental SAT solving is to exploit the progress made during the search process if similar formulae need be solved. So, instead of the learned clauses to be discarded, they are retained between the single SAT calls.

Hooker [23] presented the idea of incremental SAT solving in the context of knowledge-based reasoning. Eén and Sörensson [16] introduced the concept of *assumptions* for incremental SAT solving which fits our needs best. Assumptions can be viewed as unit clauses added to the formula. They basically represent a (partial) assignment whose literals remain set to true during the solving process. In particular, backtracking does not occur past any assumed literal.

2.5. Projection

We are interested in enumerating the models of a propositional formula *projected* onto a subset of its variables. To this end we partition the set of variables $V = X \cup Y$ into the set of *relevant variables* X and the set of *irrelevant variables* Y and write $F(X \cup Y)$ to express that F depends on the variables in $X \cup Y$. Accordingly, we decompose the assignment $\sigma = \sigma_X \cup \sigma_Y$ into its relevant part $\sigma_X : X \mapsto \mathbb{B}$ and its irrelevant part $\sigma_Y : Y \mapsto \mathbb{B}$ following the convention introduced in our earlier work on dual projected model counting [38]. The main idea of projection onto the relevant variables is to existentially quantify the irrelevant variables. The models of $F(X \cup Y)$ projected onto X are therefore

$$\mathsf{models}(\exists\, Y.\, F(X,Y)) = \{\tau : X \to \mathbb{B} \mid \text{ exists } \sigma : X \cup Y \to \mathbb{B} \text{ with } \sigma\big(F(X,Y)\big) = 1 \text{ and } \tau = \sigma_X\},$$

and enumerating all models of F without projection is therefore the special case where $Y = \emptyset$. The projection of the trail I onto the set of variables X is denoted by $\pi(I, X)$ and $\pi(F(X,Y), X) \equiv \exists\, Y\, [\, F(X,Y)\,]$.

Example 6 (*Projected Models*)**.** Consider again the formula F in Example 1. Its unprojected models are given by $\mathsf{models}(F) = \{a\, b\, c\, d,\ a\, b\, c\, \neg d,\ a\, b\, \neg c\, d,\ a\, b\, \neg c\, \neg d\}$. Its models projected onto $X = \{a, c\}$ are $a\, c$ and $a\, \neg c$.

In order to benefit from the efficient methods SAT solvers execute on CNF formulae, we transform an arbitrary formula $F(X,Y)$ into CNF by, e. g., the Tseitin transformation [64].[11] By this transformation, auxiliary variables, also referred to as *Tseitin* or *internal* variables, are introduced. The Tseitin transformation is *satisfiability-preserving*, i. e., a satisfiable formula is not turned into an unsatisfiable one and, similarly, an unsatisfiable formula is not turned into a satisfiable one. The Tseitin variables, which we denote by S, are defined in terms of the variables in $X \cup Y$, which we call *input variables*. As a consequence, for each total assignment to the variables in $X \cup Y$ there exists one single assignment to the variables in S such that the resulting assignment is a model of F, and therefore the model count is preserved. Due to the introduction of the Tseitin variables, the resulting formula $P(X,Y,S) = \mathsf{Tseitin}(F(X,Y))$ is not logically equivalent to $F(X,Y)$, i. e., $\mathsf{models}(F) \neq \mathsf{models}(P)$, and the Tseitin transformation is not *equivalence-preserving*. However, the models of $P(X,Y,S)$ projected onto the input variables are exactly the models of $F(X,Y)$, and

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

[10] After applying a factorization step, i. e., removing one $\neg c$ from the resolvent $(\neg b \vee \neg c \vee \neg c)$.

[11] It turns out that in the context of dual projected model enumeration also the Plaisted–Greenbaum transformation [51] might be used although in general it does not preserve the model count.

The models of F projected onto X are accordingly given by

$$\mathsf{models}(\exists Y, S\,[\,P(X,Y,S)\,]) = \mathsf{models}(\exists Y\,[\,F(X,Y)\,]). \tag{2}$$

2.6. Dual representation of a formula

We make use of the dual representation of a formula introduced in our earlier work [38]. Let $F(X,Y)$ and $P(X,Y,S)$ be defined as in Section 2.5, and let $N(X,Y,T) = \mathsf{Tseitin}(\neg F(X,Y))$ be a CNF representation of $\neg F$, where T denotes the set of Tseitin variables introduced by the transformation, i. e.,

$$\exists T\,[\,N(X,Y,T)\,] \equiv \neg F(X,Y). \tag{3}$$

The formulae $P(X,Y,S)$ and $N(X,Y,T)$ are a *dual representation*[12] of $F(X,Y)$.

Example 7 (*Dual Formula Representation*)**.** Let $F(X,Y) = (a \wedge \neg b) \vee (\neg a \wedge b)$ be defined over $X = \{a\}$ and $Y = \{b\}$ and suppose $\neg F(X,Y) = (a \wedge b) \vee (\neg a \wedge \neg b)$. A dual representation of $F(X,Y)$ consists of the Tseitin transformations of $F(X,Y)$ and its negation, $P(X,Y,S) = (\neg s_1 \vee a) \wedge (\neg s_1 \vee \neg b) \wedge (s_1 \vee \neg a \vee b) \wedge (\neg s_2 \vee \neg a) \wedge (\neg s_2 \vee b)(s_2 \vee a \vee \neg b) \wedge (s_1 \vee s_2)$ and $N(X,Y,T) = (\neg t_1 \vee a) \wedge (\neg t_1 \vee b) \wedge (t_1 \vee \neg a \vee \neg b) \wedge (\neg t_2 \vee \neg a) \wedge (\neg t_2 \vee \neg b) \wedge (t_2 \vee a \vee b) \wedge (t_1 \vee t_2)$, respectively, where $S = \{s_1, s_2\}$ and $T = \{t_1, t_2\}$.

For the sake of readability, we also may write F, P, and N. Notice that this representation is not unique in general. Besides that, $P(X,Y,S)$ and $N(X,Y,T)$ share the set of input variables $X \cup Y$, and $S \cap T = \emptyset$, and

$$\exists S\,[\,P(X,Y,S)\,] \equiv \neg\exists T\,[\,N(X,Y,T)\,], \tag{4}$$

In an earlier work [38] we showed that during the enumeration process a generalization of the following always holds assuming we first decide variables in X and then variables in $Y \cup S$ but never variables in T:

$$(\,\neg\exists T\,[\,N(X,Y,T)|_I\,]\,) \models (\,\exists S\,[\,P(X,Y,S)|_I\,]\,) \tag{5}$$

where I is a trail over variables in $X \cup Y \cup S \cup T$. Obviously, also

$$(\,\exists S\,[\,P(X,Y,S)|_I\,]\,) \models (\,\neg\exists T\,[\,N(X,Y,T)|_I\,]\,), \tag{6}$$

saying that whenever I can be extended to a model of P, all extensions of it falsify N. This property is a basic ingredient of our dual model shrinking method.

3. Dual reasoning for model shrinking

In a previous work, we adopted dual reasoning for obtaining partial models [38]. Basically, we executed CDCL on the formula under consideration and its negation simultaneously exploiting the fact that CDCL is biased towards detecting conflicts. Our experiments showed that dual reasoning detects short models. However, processing two formulae simultaneously turned out to be computationally expensive.

In another work [41] we propose, before taking a decision, to check whether the current (partial) assignment logically entails the formula under consideration. We present four flavors of the entailment check, some of which use a SAT oracle and rely on dual reasoning.

The method introduced in this work, instead, exploits the effectiveness of dual reasoning in detecting short partial models while avoiding both processing two formulae simultaneously and oracle calls, which might be computationally expensive. In essence, we let the enumerator find total models and shrink them by means of dual reasoning.

Assume our task is to determine the models of a formula $F(X,Y)$ over the set of relevant variables X and irrelevant variables Y projected onto X, and let $P(X,Y,S)$ and $N(X,Y,T)$ be CNF representations of F and $\neg F$, respectively, as introduced in Section 2. Obviously, Eq. (2)–Eq. (6) hold. Suppose standard CDCL is executed on P. We denote with I the trail which ranges over variables in $X \cup Y \cup S \cup T$, where S and T are the Tseitin variables occurring in P and N, respectively.

Now assume a total model I of P is found. A second SAT solver is incrementally invoked on $\pi(I, X \cup Y) \wedge N$. Since $\pi(I, X \cup Y) \models F$ and all variables in $X \cup Y$ are assigned, due to Eq. (6), a conflict in N occurs by propagating variables in T only. If conflict analysis is carried out as described in Section 2.3, the learned clause $\neg I^\star$ contains only negated assumption literals.[13] On the one hand, $\neg I^\star$ represents a cause for the conflict in N. On the other hand, due to Eq. (5), its negation $I^\star$ represents a (partial) model of F. More precisely, $I^\star$ represents all total models of F projected onto $X \cup Y$ in which the variables in $X \cup Y$ not occurring in $I^\star$ may assume any truth value.

[12] Referred to as *combined formula pair* of $F(X,Y)$ in our previous work [38].

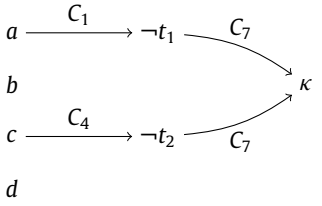
[13] See also the work by Niemetz et al. [48].

Example 8 (*Model Shrinking by Dual Reasoning*)**.** Let $F = (a \vee b) \wedge (c \vee d)$ be a propositional formula over the set of variables $V = \{a, b, c, d\}$. Without loss of generalization, suppose we want to enumerate the models of F projected onto V. Assume a total model $I = a^d\, b^d\, c^d\, d^d$ has been found. We call a second SAT solver on $N \wedge I$, where

$$N = \underbrace{(\neg t_1 \vee \neg a)}_{C_1} \wedge \underbrace{(\neg t_1 \vee \neg b)}_{C_2} \wedge \underbrace{(a \vee b \vee t_1)}_{C_3} \wedge \underbrace{(\neg t_2 \vee \neg c)}_{C_4} \wedge \underbrace{(\neg t_2 \vee \neg d)}_{C_6} \wedge \underbrace{(c \vee d \vee t_2)}_{C_6} \wedge \underbrace{(t_1 \vee t_2)}_{C_7}$$

is the Tseitin encoding of $\neg F = (\neg a \wedge \neg b) \vee (\neg c \wedge \neg d)$ with Tseitin variables $T = \{t_1, t_2\}$. The clauses C_1 to C_3 encode $(t_1 \leftrightarrow (\neg a \wedge \neg b))$, the clauses C_4 to C_6 encode $(t_2 \leftrightarrow (\neg c \wedge \neg d))$, and C_7 encodes $(t_1 \vee t_2)$.

The literals on I are considered assumed variables, annotated by, e. g., a^a, and $I = a^a\, b^a\, c^a\, d^a$. After propagating $\neg t_1$ with reason C_1 and $\neg t_2$ with reason C_4, the clause C_7 becomes empty. The current trail is $I' = a^a\, b^a\, c^a\, d^a\, \neg t_1^{C_1}\, \neg t_2^{C_4}$. We resolve C_7 with C_4 to obtain the clause $(t_1 \vee \neg c)$ which we then resolve with C_1. The resolvent is $(\neg c \vee \neg a)$ containing only assumed literals which have no reason in I', and thus cannot be resolved further. Below on the left hand side, the implication graph is visualized, and the corresponding resolution steps are depicted on the right hand side.



$$\dfrac{\dfrac{(t_1 \vee t_2) \qquad (\neg t_2 \vee \neg c)}{(t_1 \vee \neg c)} \qquad (\neg t_1 \vee \neg a)}{(\neg c \vee \neg a)}$$

The negation of the clause $(\neg c \vee \neg a)$, $c\, a$, is a counter-model of $\neg F$ and hence a model of F. In this case, it is also minimal w. r. t. the number of literals.

Note: The gain obtained by model shrinking is twofold. On the one hand, it enables the (implicit) exploration of multiple models in one pass: e. g., in Example 8, the model $c\, a$ represents four total models, namely, $a\, b\, c\, d$, $a\, b\, c\, \neg d$, $a\, \neg b\, c\, d$, and $a\, \neg b\, c\, \neg d$. On the other hand, short models result in short blocking clauses ruling out a larger part of the search space, as mentioned earlier.

4. Dual encoding of blocking clauses

Recall that in our dual model shrinking approach we rely on Eq. (4). If a blocking clause is added to P and N is not updated accordingly, then P and N do not represent the negation of each other anymore, and Eq. (4) ceases to hold. This might lead to multiple model enumerations in the further search, if dual model shrinking is applied. This issue can be remediated by adding the shrunken models disjunctively to N. The basic idea is the following. If a trail I satisfies a formula F, then it falsifies its negation $\neg F$, i. e., $F \wedge \neg I \equiv 0$ and $\neg(\neg F \wedge \neg I) = \neg F \vee I \equiv 1$. To retain $\neg F$ in CNF and ensure Eq. (4), we propose the following *dual encoding* of the blocking clauses.

We denote with $\mathsf{Tseitin}()$ the function which takes as argument an arbitrary propositional formula and returns its Tseitin transformation. For the sake of readability, we write F, P, and N as well as their indexed variants instead of $F(X \cup Y)$, $P(X \cup Y \cup S)$ and $N(X \cup Y \cup T)$. We define

$$P_0 = \mathsf{Tseitin}(F) \tag{7}$$

$$N_0 = t_0 \wedge \mathsf{Tseitin}(t_0 \leftrightarrow \neg F). \tag{8}$$

Let I_1 be a trail such that I_1 evaluates F to true, i. e., $I_1 \vdash F$. A second SAT solver *SAT* is called on $\pi(I_1, X \cup Y) \wedge N_0$, and a conflict is obtained as argued above. Assume *SAT* returns the assignment $I_1^\star \leqslant I_1$ such that $\mathit{SAT}(\pi(I_1^\star, X \cup Y), N_0) =$ UNSAT. Then $\neg\pi(I_1^\star, X)$ is added to P_0 obtaining $P_1 = P_0 \wedge \neg\pi(I_1^\star, X)$. To ensure Eq. (4), we define $N_1 = (t_0 \vee t_1) \wedge \mathsf{Tseitin}\,(t_0 \leftrightarrow \neg F) \wedge \mathsf{Tseitin}(t_1 \leftrightarrow \pi(I_1^\star, X))$, and we apply this encoding inductively as follows. At the n^{th} step, we have

$$P_n = P_0 \wedge \underbrace{\bigwedge_{i=1}^{n} \neg\pi(I_i^\star, X)} \tag{9}$$

$$N_n = \underbrace{(t_0 \vee \bigvee_{i=1}^{n} t_i)} \wedge \mathsf{Tseitin}(t_0 \leftrightarrow \neg F) \wedge \underbrace{\bigwedge_{i=1}^{n} \mathsf{Tseitin}(t_i \leftrightarrow \pi(I_i^\star, X))} \tag{10}$$

where the underlined parts denote the additions to P_0 and N_0.

Let I_{n+1} be a trail evaluating P_n to true, i.e., $I_{n+1} \vdash P_n$. We invoke *SAT* on $\pi(I_{n+1}, X \cup Y) \wedge N_n$ leading to a conflict as described above. Assume *SAT* returns $I^\star_{n+1} \leqslant I_{n+1}$, such that $SAT(\pi(I^\star_{n+1}, X \cup Y), N_n) = \text{UNSAT}$. We add $\neg\pi(I^\star_{n+1}, X)$ to P_n and update N_n accordingly. Now we have[14]

$$P_{n+1} = P_n \wedge \neg\pi(I^\star_{n+1}, X) \tag{11}$$

$$N_{n+1} = N_n \setminus \{(t_0 \vee \bigvee_{i=1}^{n} t_i)\} \wedge (t_0 \vee \bigvee_{i=1}^{n+1} t_i) \wedge \mathsf{Tseitin}(t_{n+1} \leftrightarrow \pi(I^\star_{n+1}, X)) \tag{12}$$

where $I_{i+1} \vdash P_i$ for $0 \leqslant i \leqslant n$

and $I^\star_{i+1} \leqslant I_{i+1}$ is $s.t.$ $SAT(\pi(I^\star_{i+1}, X \cup Y), N_i) = \text{UNSAT}$.

Proposition 1. *Let $F(X, Y)$ be an arbitrary propositional formula over the relevant variables X and the irrelevant variables Y. Let F and $\neg F$ be encoded into CNFs P_0 and N_0, respectively, according to Eqs. (7) and (8). If for all models found blocking clauses are added to P_0 and N_0 according to Eqs. (9) and (10), then only pairwise contradicting models are found, i.e., $\pi(I^\star_i, X)$ and $\pi(I^\star_j, X)$ are pairwise contradicting for every $i \neq j$.*

Proof. By construction, $N_n \equiv \neg F \vee \bigvee_{i=1}^{n} \pi(I^\star_i, X \cup Y)$ and, given a shrunken model $I^\star_{n+1}$ of P_n, $\pi(I^\star_{n+1}, X \cup Y) \wedge N_n \equiv 0$. Furthermore, $\pi(I^\star_{n+1}, X \cup Y) \wedge \neg F \equiv 0$. We have

$$\begin{aligned}
0 &\equiv \pi(I^\star_{n+1}, X \cup Y) \wedge (\neg F \vee \bigvee_{i=1}^{n} \pi(I^\star_i, X \cup Y)) \\
&= (\pi(I^\star_{n+1}, X \cup Y) \wedge \neg F) \vee (\pi(I^\star_{n+1}, X \cup Y) \wedge \bigvee_{i=1}^{n} \pi(I^\star_i, X \cup Y)) \\
&\equiv (\pi(I^\star_{n+1}, X \cup Y) \wedge \bigvee_{i=1}^{n} \pi(I^\star_i, X \cup Y)) \\
&\equiv (\pi(I^\star_{n+1}, X) \wedge \bigvee_{i=1}^{n} \pi(I^\star_i, X)),
\end{aligned}$$

since $I^\star_i$ contains only relevant variables. Hence, $\pi(I^\star_{n+1}, X) \wedge \pi(I^\star_i, X) \equiv 0$ for $i = 1, \ldots, n$. □

Note: Eq. (4) always holds:

$$\exists S\,[\,P_i(X, Y, S)\,] \equiv \neg\exists T\,[\,N_i(X, Y, T)\,] \quad \text{for all} \quad 0 \leqslant i \leqslant n+1.$$

Consequently, also Eqs. (5) and (6) hold:

$$(\,\neg\exists T\,[\,N_i(X, Y, T)|_I\,]\,) \models (\,\exists S\,[\,P_i(X, Y, S)|_I\,]\,) \quad \text{for all} \quad 0 \leqslant i \leqslant n+1$$

$$(\,\exists S\,[\,P_i(X, Y, S)|_I\,]\,) \models (\,\neg\exists T\,[\,N_i(X, Y, T)|_I\,]\,) \quad \text{for all} \quad 0 \leqslant i \leqslant n+1$$

However, for our usage we may use the implication in the forward direction only and write $t_i \rightarrow \pi(I^\star_i, X)$ and $t_{n+1} \rightarrow \pi(I^\star_{n+1}, X)$ in Eqs. (10) and (12) without compromising correctness for the following reason: the formula N_i is called always under I_{i+1} which falsifies all $I^\star_k$ for $0 \leqslant k \leqslant i$. Hence, $\pi(I^\star_i, X) \rightarrow t_i$ is always true.

Example 9 (*Dual Blocking Clauses*)**.** We clarify the proposed encoding by a small example and show that it prevents multiple model counts. Let our example be $F = x_1 \vee (x_2 \wedge x_3)$ and assume we have found the model $I^\star_1 = x_1$. Then

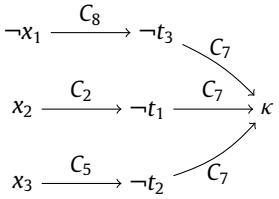
$$P_1 = (\neg s_1 \vee x_2) \wedge (\neg s_1 \vee x_3) \wedge (s_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee s_1) \wedge (\neg x_1)$$

and

$$\begin{aligned}
N_1 = {} & \underbrace{(\neg t_1 \vee \neg x_1)}_{C_1} \wedge \underbrace{(\neg t_1 \vee \neg x_2)}_{C_2} \wedge \underbrace{(t_0 \vee x_1 \vee x_2)}_{C_3} \wedge \\
& \underbrace{(\neg t_2 \vee \neg x_1)}_{C_4} \wedge \underbrace{(\neg t_2 \vee \neg x_3)}_{C_5} \wedge \underbrace{(t_2 \vee x_2 \vee x_3)}_{C_6} \wedge \\
& \underbrace{(t_1 \vee t_2 \vee t_3)}_{C_7} \wedge \underbrace{(\neg t_3 \vee x_1)}_{C_8} \wedge \underbrace{(t_3 \vee \neg x_1)}_{C_9}
\end{aligned}$$

[14] Notice that here, with a little abuse of notation, by "$N_n \setminus \{C\}$" we refer to the formula resulting from dropping clause C from formula N_n.

where the unit clause $(\neg x_1)$, the literal t_3 in C_7 as well as the clauses C_8 and C_9, all emphasized in blue, denote the corresponding additions to P_0 and N_0. If now we find a total model $I_2 = \neg x_1\, x_2\, x_3$, we obtain a conflict in N_1 by unit propagating variables t_1, t_3, and t_3 only. The conflicting clause is $(t_1 \vee t_2 \vee t_3)$. The implication graph is depicted below on the left hand side, the corresponding resolution steps for conflict analysis below on the right hand side.



$$\dfrac{\dfrac{\dfrac{\dfrac{(t_1 \vee t_2 \vee t_3) \quad (\neg t_2 \vee \neg x_3)}{(t_1 \vee t_3 \vee \neg x_3)} \quad (\neg t_1 \vee \neg x_2)}{(t_3 \vee \neg x_3 \vee \neg x_2)} \quad (\neg t_3 \vee x_1)}{(\neg x_3 \vee \neg x_2 \vee x_1)}$$

Conflict analysis returns the clause $(\neg x_3 \vee \neg x_2 \vee x_1)$, which, after being added to P, blocks the model $\neg x_1\, x_2\, x_3$, which does not overlap with the previously found model x_1.

5. Projected model enumeration without repetition

We are given a propositional formula $F(X, Y)$ over the set of relevant variables X and the set of irrelevant variables Y, and our task is to enumerate its models projected onto the variables in X. Let $P(X, Y, S)$ and $N(X, Y, T)$ be a dual representation of F according to Section 2. Obviously, Eq. (2)–Eq. (6) hold. Now we call a CDCL-based SAT solver on P. Whenever it finds a total model I of P, it is shrunken by dual reasoning obtaining $I^\star$ which also satisfies P. The decision level of $I^\star$ might be significantly smaller than the one of I, and backtracking to decision level $b = \delta(\neg\pi(I^\star, X)) - 1$ mimics non-chronological backtracking in CDCL. Notice that $I^\star$ is used solely for determining the backtracking level. The blocking clause added to P consists of all decisions with decision level smaller or equal to $b + 1$ and propagates after backtracking.

In Fig. 3, we consider the case with permanent learning of the blocking clauses. Let the first SAT solver execute standard CDCL on P and let I denote its trail. Obviously it finds only total models of P. Due to Eq. (2), these models satisfy F, too. Now assume a (total) model I of P is found. A second SAT solver *SAT* is incrementally invoked on $\pi(I, X \cup Y) \wedge N$ with the aim to shrink I obtaining $I^\star$ as described in Section 3.

Let b denote the decision level of $\pi(\neg I^\star, X)$ and ℓ be the literal in $\pi(\neg I^\star, X)$ with decision level b. We now add the clause $\pi(\neg\mathsf{decs}(I^\star), X)$ to P and backtrack to decision level $b - 1$. Notice that $\pi(\neg\mathsf{decs}(I^\star), X)$ acts in P as a blocking clause and must not be deleted anytime which might blow up P and slow down the first SAT solver. Moreover, the dual encoding of the blocking clause according to Section 4 ensures Eq. (4) on which our method relies.

In Section 5.1, we present the main function EnumerateIrredundant. Unit propagation (Section 5.2) and the schema for conflict analysis (Section 5.3) are the same as in CDCL for SAT.

5.1. Main algorithm

The function EnumerateIrredundant in Fig. 3 describes the main algorithm. (Black rows 1–18 and 27–28 represent standard CDCL with projection returning a model if the formula under consideration is satisfiable and the empty clause otherwise, blue rows 19–26 the rest of the algorithm.)

Initially, the trail I is empty, the target DNF M is 0, and all variables are unassigned, i. e., assigned decision level ∞. Unit propagation is executed until either a conflict occurs or all variables are assigned a value (line 7).

If a conflict occurs at decision level zero, the search space has been processed exhaustively, and the enumeration terminates (lines 8–11). If a conflict occurs at a decision level higher than zero, conflict analysis is executed (line 13).

If no conflict occurs and all variables are assigned, a total model has been found (line 15). If no relevant decisions are left on the trail I, the relevant search space has been processed exhaustively, the found model is output and the search terminates (lines 17–18). If I contains a relevant decision, the found model is shrunken (line 20) by means of dual reasoning as described in Section 3. It is blocked, and the last relevant decision literal is flipped (lines 22–26). If no conflict occurs and not all variables are assigned, a decision is taken (line 28), where the variables in X are prioritized over the variables in $Y \cup S$ to avoid enumerating models which only differ in irrelevant and Tseitin variables.

5.2. Unit propagation

Unit propagation is described by the function PropagateUnits in Fig. 4. It takes as input the formula F, the trail I, and the decision level function δ. If a clause $C \in F$ is unit under I, its unit literal ℓ is propagated, i. e., I is extended by ℓ (line 2). Propagated literals are assigned at the current decision level (line 3) as is usual in modern CDCL-based SAT solvers. If the resulting trail falsifies some clause $D \in F$, this clause is returned (lines 4–5). Otherwise the function returns the empty clause 0 (line 6).

Input: formulae $P(X,Y,S)$ and $N(X,Y,T)$ s. t.
$\exists S\,[\,P(X,Y,S)\,] \equiv \neg\exists T\,[\,N(X,Y,T)\,]$,
set of variables $X \cup Y \cup S \cup T$

Output: DNF representation of $\pi(P,X)$

```
EnumerateIrredundant ( P, N )        // P_0 = CNF ( F )
                                     // N_0 = t_0 ∧ CNF ( t_0 ↔ ¬F )
 1  I := ε
 2  δ[ V ↦ ∞ ]
 3  M := 0
 4  i := 0
 5  forever do
 6    i := i + 1
 7    C := PropagateUnits ( P, I, δ )
 8    if C ≠ 0 then
 9      c := δ(C)
10      if c = 0 then
11        return M
12      else
13        AnalyzeConflict ( P, I, C, δ )
14    else
15      if all variables in X ∪ Y ∪ S are assigned then
16        // I is total model of P and F
17        if V(decs(I)) ∩ X = ∅ then
18          return M ∨ π(I, X)
19        else
20          I* := CSet ( N, π(I, X ∪ Y) )
21          // I* is model of π(F, X ∪ Y) and conflict set of I w. r. t. N
22          P := P ∧ ¬π(decs(I*), X)
23          N := N \ {(t_0 ∨ ⋁_{j=1}^{i-1} t_j)}∧
                        (t_0 ∨ ⋁_{j=1}^{i} t_j) ∧ CNF ( t_i ↔ π(decs(I*), X) )
24          M := M ∨ π(I*, X)
25          b := δ(¬π(I*, X))
26          Backtrack ( I, b − 1 )
27      else
28        Decide ( P, I, δ )
```

Fig. 3. Irredundant model enumeration. The black lines 1–18 and 27–28 describe CDCL returning a model if one is found and the empty clause otherwise. Notice that for decisions, variables in X are prioritized over variables in $(Y \cup S)$ to avoid multiple enumeration of projected models. Similarly, in line 17 it suffices to check whether no relevant decision literals are left on the trail. The blue part, i.e., lines 19–26, represents the extension to model enumeration. A second SAT solver is called incrementally on N assuming the literals on $\pi(I, X \cup Y)$. A conflict occurs by unit propagation only, and $\pi(I^\star, X \cup Y)$ is a (partial) model of F. It is encoded as a dual blocking clause, and P and N are updated accordingly.

5.3. Conflict analysis

Conflict analysis is described by the function AnalyzeConflict in Fig. 4. It takes as input the formula F, the trail I, the conflicting clause C, and the decision level function δ. A clause D is learned as described in Section 7 and added to F (lines 1–2). The second highest decision level j in D is determined (lines 3–4), and the enumerator backtracks (non-chronologically) to decision level j. Backtracking involves unassigning all literals with decision level higher than j (lines 5–7). After backtracking, the clause D becomes unit with unit literal ℓ, which is propagated and assigned decision level j (lines 8–9).

```
PropagateUnits ( F, I, δ )
1  while some C ∈ F is unit (ℓ) under I do
2    I := I ℓ
3    δ(ℓ) := δ(I)
4    for all clauses D ∈ F containing ¬ℓ do
5      if I(D) = 0 then return D
6  return 0

AnalyzeConflict ( F, I, C, δ )
1    D := Learn ( I, C )
2    F := F ∧ D
3    ℓ := literal in D at decision level δ(I)
4    j := δ(D \ {ℓ})
5  for all literals k ∈ I with decision level > j do
6    assign k decision level ∞
7    remove k from I
8  I := I ℓ
9  δ(ℓ) := j
```

Fig. 4. The function PropagateUnits implements unit propagation in F. The unit literal ℓ is assigned the decision level of I. If some clause $D \in F$ containing the complement of ℓ becomes falsified, PropagateUnits returns D. Otherwise it returns the empty clause 0 indicating that no conflict has occurred. The function AnalyzeConflict is called whenever a clause $C \in F$ becomes empty under the current assignment. It learns a clause D starting with the conflicting clause C. The solver then backtracks to the second highest decision level j in D upon which D becomes unit with unit literal ℓ, and propagates ℓ.

6. Formalizing projected irredundant model enumeration

In this section, we provide a formalization of our algorithm presented in Section 5. Let $F(X, Y)$ be a formula defined onto the set of relevant (input) variables X and the set of irrelevant (input) variables Y, and assume our task is to enumerate its models projected onto X.

Our formalization works on a dual representation of F, given by $P(X, Y, S)$ and $N(X, Y, T)$, as introduced in Section 2.6. So, $P(X, Y, S)$ and $N(X, Y, T)$ are defined over the same sets of relevant variables X and irrelevant variables Y as well as the disjoint sets of variables S and T, respectively, which are defined in terms of the variables in $X \cup Y$. Recall that Eq. (2)–Eq. (6) hold. We start by sketching the enumeration process. After presenting our calculus, we show its working by means of an example before providing a correctness proof.

The process works as follows. Let I denote the current trail. Unit propagation is carried out as long as $P|_I$ contains a unit literal (rule Unit). If all variables in $X \cup Y \cup S$ are assigned and no conflict has occurred, a total model of P has been found. In case there is no decision left on I, its projection onto X is enumerated, and we are done (End1). Otherwise, the model I is shrunken and blocked by means of the dual blocking clause encoding (Back1). If a conflict occurred and there is no decision left on I, the process terminates (End0). Otherwise, conflict analysis is carried out and backtracking occurs (Back0). If after executing exhaustive unit propagation there are still unassigned variables in $X \cup Y \cup S$, a decision need be taken. We are interested in the models projected onto X. To avoid detecting models which differ only in variables $Y \cup S$, we first decide variables in X (DecX), before deciding variables in $Y \cup S$ (DecYS).

6.1. Calculus

We formalize the algorithm presented in Section 5 as a state transition system with transition relation $\rightsquigarrow_{\mathsf{EnumIrred}}$. Non-terminal states are described by tuples (P, N, M, I, δ). The third element, M, is a DSOP formula over variables in X. The fourth element, I, denotes the trail defined over variables in $X \cup Y \cup S \cup T$, and δ denotes the decision level function. The initial state is $(P_0, N_0, 0, \varepsilon, \delta_0)$, where P_0 and N_0 denote the initial CNF representations of F and $\neg F$, respectively, ε denotes the empty trail, and $\delta_0 \equiv \infty$. The terminal state is given by a DSOP formula M, which is equivalent to the projection of P onto X. The transition relation $\rightsquigarrow_{\mathsf{EnumIrred}}$ is the union of transition relations $\rightsquigarrow_{\mathsf{R}}$, where R is either End1, End0, Unit, Back1, Back0, DecX, or DecYS. The rules are listed in Fig. 5.

End1. All variables are assigned and no conflict in P occurred, hence the trail I is a total model of P. It contains no relevant decision indicating that the relevant search space has been processed exhaustively. The model projected onto X is added to M, and the search terminates. It is sufficient to check for relevant decisions, since flipping an irrelevant one would result in detecting redundant models projected onto X. However, due to the addition of blocking clauses, a conflict would occur, and checking for relevant decisions essentially saves work.

End1: $(P, N, M, I, \delta) \rightsquigarrow_{\mathsf{End1}} M \vee m$ if $P|_I \neq 0$ and $(X \cup Y \cup S) - I = \emptyset$ and $V(\mathsf{decs}(I)) \cap X = \emptyset$ and $m \stackrel{\text{def}}{=} \pi(I, X)$

End0: $(P, N, M, I, \delta) \rightsquigarrow_{\mathsf{End0}} M$ if exists $C \in P$ with $C|_I = 0$ and $\delta(C) = 0$

Unit: $(P, N, M, I, \delta) \rightsquigarrow_{\mathsf{Unit}} (P, N, M, I\ell^C, \delta[\ell \mapsto a])$ if $P|_I \neq 0$ and exists $C \in P$ with $\{\ell\} = C|_I$ and $a \stackrel{\text{def}}{=} \delta(I)$

Back1: $(P, N, M, I, \delta) \rightsquigarrow_{\mathsf{Back1}} (P \wedge B, O, M \vee m, J\ell^B, \delta[K \mapsto \infty][\ell \mapsto b])$ if $(X \cup Y \cup S) - I = \emptyset$ and exists $I^\star \leqslant \pi(I, X \cup Y)$ with $JK = I$ such that $N \wedge I^\star \vdash_1 0$ and $m \stackrel{\text{def}}{=} \pi(I^\star, X)$ and $B \stackrel{\text{def}}{=} \neg\mathsf{decs}(m)$ and $b + 1 \stackrel{\text{def}}{=} \delta(B) = \delta(m)$ and $\ell \in B$ and $\ell|_K = 0$ and $b = \delta(B \setminus \{\ell\}) = \delta(J)$ and $O = \mathsf{Tseitin}(N \vee \neg B)$

Back0: $(P, N, M, I, \delta) \rightsquigarrow_{\mathsf{Back0}} (P \wedge D^r, N, M, J\ell^D, \delta[K \mapsto \infty][\ell \mapsto j])$ if exists $C \in P$ and exists D with $JK = I$ and $C|_I = 0$ and $\delta(C) = \delta(D) > 0$ such that $\ell \in D$ and $\neg\ell \in \mathsf{decs}(I)$ and $\neg\ell|_K = 0$ and $P \models D$ and $j \stackrel{\text{def}}{=} \delta(D \setminus \{\ell\}) = \delta(J)$

DecX: $(P, N, M, I, \delta) \rightsquigarrow_{\mathsf{DecX}} (P, N, M, I\ell, \delta[\ell \mapsto d])$ if $P|_I \neq 0$ and $\mathsf{units}(P|_I) = \emptyset$ and $\delta(\ell) = \infty$ and $d \stackrel{\text{def}}{=} \delta(I) + 1$ and $V(\ell) \in X$

DecYS: $(P, N, M, I, \delta) \rightsquigarrow_{\mathsf{DecYS}} (P, N, M, I\ell, \delta[\ell \mapsto d])$ if $P|_I = 0$ and $\mathsf{units}(P|_I) = \emptyset$ and $\delta(\ell) = \infty$ and $d \stackrel{\text{def}}{=} \delta(I) + 1$ and $V(\ell) \in Y \cup S$ and $X - I = \emptyset$

Fig. 5. Rules for projected model enumeration without repetition. States are represented as tuples (P, N, M, I, δ). The formulae $P(X, Y, S)$ and $N(X, Y, T)$ are a dual representation of the formula $F(X.Y)$ whose models projected onto X are sought. These models are recorded in the initially empty DNF M. The last two elements, I and δ, denote the current trail and decision level function, respectively. If a model is found or a conflict encountered and the search space has been processed exhaustively, the search terminates (rules End1 and End0). Otherwise, either the model is shrunken and a dual blocking clause is added (rules Back1) or conflict analysis is executed followed by non-chronological backtracking (rule Back0). If the residual of P under the current trail IJ contains a unit literal, it is propagated (rule Unit). If none of the mentioned preconditions are met, a decision is taken. Relevant variables are prioritized (rule DecX) over irrelevant and internal ones (rule DecYS).

End0. A conflict at decision level zero has occurred indicating that the search space has been processed exhaustively. The search terminates leaving M unaltered. We need to make sure no decision is left on the trail, which in particular includes the irrelevant ones. The reason is that after flipping any decision — in particular also irrelevant and internal ones — the resulting trail might be extended to a model of P.

Unit. No conflict in P occurred, and a clause in P is unit under I. Its unit literal ℓ is propagated and assigned the current decision level.

Back1. All variables are assigned, no conflict in P occurred, and the trail I is a total model of P. It is shrunken as described in Section 3 obtaining $I^\star$. The projection m of $I^\star$ onto X is added to M. The clause B consisting of the negated decision literals of m is added as a blocking clause to P. Its negation $\neg B$ is added disjunctively to N, which is transformed back into CNF by means of the Tseitin transformation.[15] The solver backtracks to the second highest decision level in B and propagates ℓ at the current decision level, i. e., basically flips the relevant decision literal with highest decision level.

Back0. The current trail falsifies a clause in P at a decision level greater than zero indicating that the search space has not yet been processed exhaustively. Conflict analysis returns a clause D implied by P, which is added to P and marked as redundant. The solver backtracks to the second highest decision level j in D. The learned clause D becomes unit and its unit literal ℓ is propagated at decision level j. In contrast to End1, every decision literal need be flipped, which particularly applies to irrelevant and internal decision literals.

[15] Notice that although not stated explicitly in favor of simplifying the presentation, the dual blocking clause encoding introduced in Section 4 may be used (see lines 22–23 of algorithm EnumerateIrredundant in Fig. 3).

Step	Rule	I	$P\|_I$	N	M
0		ε	P_0	N_0	0
1	DecX	a^d	$(b \vee d) \wedge (b \vee \neg d)$	N_0	0
2	DecX	$a^d\, c^d$	$(b \vee d) \wedge (b \vee \neg d)$	N_0	0
3	DecYS	$a^d\, c^d\, b^d$	1	N_0	0
4	DecYS	$a^d\, c^d\, b^d\, d^d$	1	N_0	0
5	Back1	$\neg a^{B_1}$	$(c) \wedge (\neg c) \wedge (b \vee d) \wedge (b \vee \neg d)$	N_1	a
6	Unit	$\neg a^{B_1}\, c^{C_1}$	$() \wedge (b \vee d) \wedge (b \vee \neg b)$	N_1	a
7	End0				a

Fig. 6. Execution trace for $F = (a \vee c) \wedge (a \vee \neg c) \wedge (b \vee d) \wedge (b \vee \neg d)$ defined over the set of relevant variables $X = \{a, c\}$ and the set of irrelevant variables $Y = \{b, d\}$ (see also Examples 1 and 6).

DecX. No conflict has occurred, the residual of P under I contains no units, and there is an unassigned relevant literal ℓ. The current decision level is incremented to d, the literal ℓ is decided and assigned to decision level d.

DecYS. No conflict has occurred, and the residual of P under I contains no units. All relevant literals are assigned, and there is an unassigned irrelevant or internal literal ℓ. The current decision level is incremented to d, ℓ is decided and assigned decision level d.

6.2. Example

The working of our calculus is shown by means of an example. Consider again Examples 1 and 6. We have

$$F = \underbrace{(a \vee c)}_{C_1} \wedge \underbrace{(a \vee \neg c)}_{C_2} \wedge \underbrace{(b \vee d)}_{C_3} \wedge \underbrace{(b \vee \neg d)}_{C_4}$$

and assume the set of relevant variables is $X = \{a, c\}$ and the set of irrelevant variables is $Y = \{b, d\}$. The formula F is already in CNF, therefore we define $P_0 = F$ and accordingly $S_0 = \emptyset$. For its negation

$$\neg F = (\neg a \wedge \neg c) \vee (\neg a \wedge c) \vee (\neg b \wedge \neg d) \vee (\neg b \wedge d)$$

we define

$$\begin{aligned} N_0 = {} & \underbrace{(\neg t_1 \vee \neg a)}_{D_1} \wedge \underbrace{(\neg t_1 \vee \neg c)}_{D_2} \wedge \underbrace{(a \vee c \vee t_1)}_{D_3} \wedge \\ & \underbrace{(\neg t_2 \vee \neg a)}_{D_4} \wedge \underbrace{(\neg t_2 \vee c)}_{D_5} \wedge \underbrace{(a \vee \neg c \vee t_2)}_{D_6} \wedge \\ & \underbrace{(\neg t_3 \vee \neg b)}_{D_7} \wedge \underbrace{(\neg t_3 \vee \neg d)}_{D_8} \wedge \underbrace{(b \vee d \vee t_3)}_{D_9} \wedge \\ & \underbrace{(\neg t_4 \vee \neg b)}_{D_{10}} \wedge \underbrace{(\neg t_4 \vee d)}_{D_{11}} \wedge \underbrace{(b \vee \neg d \vee t_4)}_{D_{12}} \wedge \\ & \underbrace{(t_1 \vee t_2 \vee t_3 \vee t_4)}_{D_{13}} \end{aligned}$$

with the set of internal variables $T_0 = \{t_1, t_2, t_3, t_4\}$. Assume a lexicographic ordering of the input variables, i. e., $a \succ_{lex} b \succ_{lex} c \succ_{lex} d$, and assume we choose the decision variable according to this ordering. The execution steps are depicted in Fig. 6.

Step 0: The initial state is given by the empty trail ε, the CNF formulae P_0 and N_0, and the empty DNF formula 0.

Step 1: The formula P_0 contains no units and there are unassigned relevant variables. The preconditions of rule DecX are met, and decision a is taken.

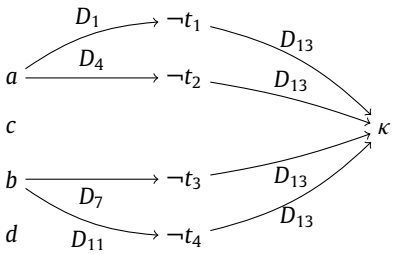
Step 2: No conflict occurred, $P_0|_I$ contains no units, and there are unassigned relevant variables. The preconditions of rule DecX are met, and c is decided.

Step 3: No conflict occurred, and $P_0|_I$ contains no units. All relevant variables are assigned and there are unassigned irrelevant variables. The preconditions of rule DecYS are met, and decision b is taken. Notice that I already satisfies P_0, but the solver is not able to detect this fact.

Step 4: Again, the preconditions of rule DecYS are met, and decision d is taken.

Step 5: No conflict occurred and all variables are assigned, hence I is a model of P_0. It is shrunken following the procedure described in Section 3. We call a SAT solver on $N_0 \wedge I$ assuming the literals on I. A conflict in N_0 occurs by

propagation of variables in T_0, and conflict analysis provides us with the shrunken model $a\,b$ of F. The resulting implication graph and trail are as follows:



$$I = a^a\, c^a\, b^a\, d^a\, \neg t_1{}^{D_1}\, \neg t_2{}^{D_4}\, \neg t_3{}^{D_7}\, \neg t_4{}^{D_{10}}$$

The conflicting clause is D_{13}. For conflict analysis, we resolve D_{13} with D_{10} and the resolvent with D_7 followed by resolution with D_4 and D_1. The obtained clause $(\neg b \vee \neg a)$ contains only assumed literals. The assumptions c and d do not participate in the conflict and therefore do not occur in the resulting clause. Below, the resolution steps are visualized.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(t_1 \vee t_2 \vee t_3 \vee t_4) \qquad (\neg t_4 \vee \neg b)}{(t_1 \vee t_2 \vee t_3 \vee \neg b)} \qquad (\neg t_3 \vee \neg b)}{(t_1 \vee t_2 \vee \neg b)} \qquad (\neg t_2 \vee \neg a)}{(t_1 \vee \neg b \vee \neg a)} \qquad (\neg t_1 \vee \neg a)}{(\neg b \vee \neg a)}$$

The negation of $(\neg b \vee \neg a)$ is $I^\star = a\,b \leqslant I$ we are looking for. The first model is $m_1 = \pi(I^\star, X) = a$ and accordingly $M_1 = M_0 \vee m_1$. Furthermore, we have $B_1 = \neg\mathsf{decs}(m_1) = (\neg a)$, hence

$$\begin{aligned}
P_1 &= P_0 \wedge \underbrace{(\neg a)}_{B_1} \quad \text{and} \\
N_1 &= N_0 \vee \underbrace{(a)}_{\neg B_1} \\
&= N_0 \setminus \{(\bigvee_{j=1}^{4} t_j)\} \wedge (\bigvee_{j=1}^{5} t_j) \wedge (t_5 \leftrightarrow a) \\
&= (\bigwedge_{i=1}^{12} D_i) \wedge \underbrace{(\neg t_5 \vee a)}_{D_{14}} \wedge \underbrace{(\neg a \vee t_5)}_{D_{15}} \wedge \underbrace{(t_1 \vee t_2 \vee t_3 \vee t_4 \vee t_5)}_{D_{16}}
\end{aligned}$$

where $D_{14} \wedge D_{15} = (t_5 \leftrightarrow a)$ is the Tseitin transformation of m_1. The clause $\neg B_1$ is added disjunctively to N_0. To retain N in CNF, $\neg B_1$ is encoded as $(t_5 \leftrightarrow \neg B_1)$, t_5 is added to D_{13} resulting in D_{16} and $T_1 = T_0 \cup \{t_5\} = \{t_1, t_2, t_3, t_4, t_5\}$ as described in Section 4. The clause B_1 acts in P as blocking clause. The solver backtracks to decision level zero and propagates $\neg a$ with reason B_1.

Step 6: The formula $P_1|_l$ contains two units, $C_1|_l = (c)$ and $C_2|_l = (\neg c)$. The literal c is propagated with reason C_1.

Step 7: The trail falsifies C_2 and the current decision level is zero. The preconditions of rule End0 are met and the search terminates without altering $M = a$, which represents exactly the models of F projected onto X, namely $a\,c$ and $a\,\neg c$.

6.3. Proofs

Our proofs are based on the ones provided for our work addressing chronological CDCL for model counting [40], which in turn rely on the proof of correctness we provided for chronological CDCL [39]. The method presented here mainly differs from the former in the following aspects: The total models found are shrunken by means of dual reasoning. It adopts non-chronological CDCL instead of chronological CDCL and accordingly makes use of blocking clauses, which affects the ordering of the literals on the trail. In fact, unlike in chronological CDCL, the literals on the trail are ordered in ascending order with respect to their decision level, which simplifies not only the rules but also the proofs. Projection in turn adds complexity to some invariants. In some aspects our proofs are similar to or essentially the same as those in our former proofs [39,40]. However, they are fully worked out to keep them self-contained.

In order to prove the correctness of our method, we make use of the invariants listed in Fig. 7. Invariant InvDualPN in essence is Eq. (4). It ensures that the shrunken model is again a model of P projected onto the input variables, stating that

InvDualPN:	$\exists S\,[\,P(X,Y,S)\,] \equiv \neg\exists T\,[\,N(X,Y,T)\,]$
InvDecs:	$\forall b \in \{1,\ldots,\delta(I)\}\,\exists!\ell \in \mathsf{decs}(I)\,[\delta(\ell) = b]$
InvImplIrred:	$\forall n \in \mathbb{N}\,.\,P \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$
InvDSOP:	M is a DSOP

Fig. 7. Invariants for projected model enumeration without repetition.

P and N projected onto the input variables $X \cup Y$ are each other's negation. Intuitively, Invariant InvDualPN holds because the found models are blocked in P and added to its negation N. Invariants InvDecs and InvImplIrred equal Invariants (2) and (3) in our proofs of correctness of chronological CDCL [39] and model counting by means of chronological CDCL [40]. Invariant InvImplIrred differs from the latter in that we need not consider the negation of the DNF M explicitly. The negation of M is exactly the conjunction of the blocking clauses associated with the found models, and these are added to P. Invariant InvImplIrred is needed to show that the literal propagated after backtracking is implied by the resulting trail. Its reason is either a blocking clause (rule Back1) or a clause learned by means of conflict analysis (rule Back0).

Our proof is split into several parts. We start by showing that the invariants listed in Fig. 7 hold in non-terminal states (Section 6.3.1). Then we prove that our procedure always makes progress (Section 6.3.2) before showing its termination (Section 6.3.3). We conclude the proof by showing that every total model is found exactly once and that all total models are detected, i. e., that upon termination $M \equiv \pi(P, X)$ holds (Section 6.3.4).

6.3.1. Invariants in non-terminal states

Proposition 2. *Invariants InvDualPN, InvDecs, InvDSOP, and InvImplIrred hold in non-terminal states.*

Proof. The proof is carried out by induction over the number of rule applications. Assuming Invariants InvDualPN to InvImplIrred hold in a non-terminal state $(P,\ N,\ M,\ I,\ \delta)$, we show that they are met after the transition to another non-terminal state for all rules.

Unit

Invariant InvDualPN: Neither P nor N are altered, hence Invariant InvDualPN holds after the application of rule Unit.

Invariant InvDecs: The trail I is extended by a literal ℓ. We need to show that ℓ is not a decision literal. Only the case where $a > 0$ need be considered, since at decision level zero all literals are propagated. There exists a clause $C \in P$ s. t. $C|_I = \{\ell\}$. Now, $a = \delta(I)$, i. e., there is already a literal $k \neq \ell$ on I with $\delta(k) = a$. From this it follows that ℓ is not a decision literal. The decisions remain unchanged, and Invariant InvDecs holds after applying rule Unit.

Invariant InvImplIrred: Due to $C|_I = \{\ell\}$, we have $P \wedge \mathsf{decs}_{\leqslant n}(I) \models \neg(C \setminus \{\ell\})$. Since $C \in P$, also $P \wedge \mathsf{decs}_{\leqslant n}(I) \models C$. Modus ponens gives us $P \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$. Hence, $P \wedge \mathsf{decs}_{\leqslant n}(I\ell) \models I\ell_{\leqslant n}$, and Invariant InvImplIrred holds after executing rule Unit.

Invariant InvDSOP: Due to the premise, M is a DSOP. It is not altered by rule Unit and after its application is therefore still a DSOP.

Back1

Invariant InvDualPN: We have $\exists S\,[\,P(X,Y,S)\,] \equiv \neg\exists T\,[\,N(X,Y,T)\,]$ and we need to show $\exists S\,[\,(P \wedge B)(X,Y,S)\,] \equiv \neg\exists T\,[\,O(X,Y,T)\,]$, where $B = \neg\mathsf{decs}(m)$ and $O = \mathsf{Tseitin}(N \vee \neg B)$ and $m = \pi(I^\star, X)$ is a model of P projected onto X. Since we have that $\exists T\,[\,O(X,Y,T)\,] \equiv \exists T\,[\,(N \vee \neg B)(X,Y,T)\,]$ and furthermore $\neg\exists T\,[\,(N \vee \neg B)(X,Y,T)\,] \equiv \forall T\,[\,(\neg N \wedge B)(X,Y,T)\,]$, we reformulate the claim as $\exists S\,[\,(P \wedge B)(X,Y,S)\,] \equiv \forall T\,[(\neg N \wedge B)(X,Y,T)\,]$. Together with $\exists S\,[\,P(X,Y,S)\,] \equiv \forall T\,[\,\neg(N(X,Y,T))]$ and observing that B contains no variable in $S \cup T$, the claim holds.

Invariant InvDecs: We show that the decisions remaining on the trail are unaffected and that no new decision is taken, i. e., ℓ in the post state is not a decision. It is sufficient to consider the case where $\delta(I) > 0$. Now, $J = I_{\leqslant b}$ by the definition of J, and the decisions on J are not affected by rule Back1. We have $\delta(B \setminus \{\ell\}) = b = \delta(J)$ and $\delta(B) = b + 1$. Since relevant decisions are prioritized, also $B = \neg\mathsf{decs}_{\leqslant b+1}(\pi(I,X)) = \neg\mathsf{decs}_{\leqslant b+1}(I)$. By the induction hypothesis, there exists exactly one decision literal for each decision level and in particular in B. Since $\ell \in B$, we have $\neg\ell \in \mathsf{decs}(I)$. Precisely, $\neg\ell \in K$, and $\neg\ell$ is unassigned upon backtracking. Due to the definition of B, there exists a literal $k \in B$ where $k \neq \ell$ such that $\delta(k) = b$, i. e., $k \in J$, hence k precedes ℓ on the resulting trail. By the definition of the blocks on the trail, ℓ is not a decision literal. Since the decisions on J are unaffected, as argued above, Invariant InvDecs is met.

Invariant InvImplIrred: We need to show that $P \wedge \mathsf{decs}_{\leqslant n}(J\,\ell) \models (J\,\ell)_{\leqslant n}$ for all n. First notice that the decision levels of the literals in J do not change by applying rule Back1. Only the decision level of the variable of ℓ is decremented from $b+1$ to b. It also stops being a decision. Since $\delta(J\,\ell) = b$, we can assume $n \leqslant b$. Observe that $P \wedge \mathsf{decs}_{\leqslant n}(J\,\ell) \equiv P \wedge \mathsf{decs}_{\leqslant n}(J)$, since ℓ is not a decision in $J\,\ell$ and $I_{\leqslant b} = J$ and thus $I_{\leqslant n} = J_{\leqslant n}$ by definition. Now the induction hypothesis is applied and we get $P \wedge \mathsf{decs}_{\leqslant n}(J\,\ell) \models I_{\leqslant n}$. Again using $I_{\leqslant n} = J_{\leqslant n}$ this almost closes the proof except that we are left to prove

$P \wedge \mathsf{decs}_{\leqslant b}(J\,\ell) \models \ell$ as ℓ has decision level b in $J\,\ell$ after applying the rule and thus ℓ disappears in the proof obligation for $n < b$. To see this notice that $P \wedge \neg B \models I_{\leqslant b+1}$ using again the induction hypothesis for $n = b+1$, and recalling that relevant decisions are prioritized, i. e., $I_{\leqslant b+1}$ contains only relevant decisions, and $\neg B = \mathsf{decs}(\pi(I^\star, X)) = \mathsf{decs}_{\leqslant b+1}(I)$. This gives $P \wedge \neg\mathsf{decs}_{\leqslant b}(J) \wedge \neg\ell \models I_{\leqslant b+1}$ and thus $P \wedge \neg\mathsf{decs}_{\leqslant b}(J) \wedge \neg I_{\leqslant b+1} \models \ell$ by conditional contraposition. Therefore, Invariant InvImplIrred holds.

Invariant InvDSOP: We assume that M is a DSOP and need to show that $M \vee m$ is also a DSOP. Due to the use of the dual blocking clause encoding, Proposition 1 holds, and Invariant InvDSOP is met after executing Back1.

Back0

Invariant InvDualPN: We have $\exists S\,[\,P(X,Y,S)\,] \equiv \neg\exists T\,[\,N(X,Y,T)\,]$, and we need to show that $\exists S\,[\,(P \wedge D)(X,Y,S)\,] \equiv \neg\exists T\,[\,N(X,Y,T)\,]$. By the premise, $P \models D$, hence $P \wedge D \equiv P$. Now $\exists S\,[\,(P \wedge D)(X,Y,S)\,] \equiv \exists S\,[\,P(X,Y,S)\,] \equiv \neg\exists T\,[\,N(X,Y,T)\,]$, and Invariant InvDualPN holds.

Invariant InvDecs: We have $J \leqslant I$, hence the decisions on J remain unaltered. Now we show that ℓ is not a decision literal. As in the proof for rule Unit, it is sufficient to consider the case where $j > 0$. There exists a clause D where $P \models D$ such that $\delta(D) > 0$ and a literal $\ell \in D$ for which $\ell|_K = 0$ and $\neg\ell \in K$, hence ℓ is unassigned during backtracking. Furthermore, there exists a literal $k \in D$ where $k \neq \ell$ and such that $\delta(k) = j$ which precedes ℓ on the trail $J\,\ell$. Therefore, following the argument in rule Unit, the literal ℓ is not a decision literal. Since the decisions remain unchanged, Invariant InvDecs holds after applying rule Back0.

Invariant InvImplIrred: Let n be arbitrary but fixed. Before executing rule Back0, we have $P \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$. We need to show that $P \wedge \mathsf{decs}_{\leqslant n}(J\,\ell) \models (J\,\ell)_{\leqslant n}$. Now, $I = J\,K$ and $J < I$, i. e., $P \wedge \mathsf{decs}_{\leqslant n}(J) \models J_{\leqslant n}$. From $j = \delta(D \setminus \{\ell\}) = \delta(J)$ we get $D|_J = \{\ell\}$. On the one hand, $P \wedge \mathsf{decs}_{\leqslant n}(J) \models \neg(D \setminus \{\ell\})$, and on the other hand $P \wedge \mathsf{decs}_{\leqslant n}(J) \models D$. Therefore, by modus ponens, $P \wedge \mathsf{decs}_{\leqslant n}(J) \models \ell$. Since ℓ is not a decision literal, as shown above, $P \wedge \mathsf{decs}_{\leqslant n}(J) \equiv P \wedge \mathsf{decs}_{\leqslant n}(J\,\ell)$ and $P \wedge \mathsf{decs}_{\leqslant n}(J\,\ell) \models J\,\ell$, and Invariant InvImplIrred holds after applying rule Back0.

Invariant InvDSOP: The DSOP M remains unaltered, and Invariant InvDSOP still holds after executing rule Back0.

DecX

Invariant InvDualPN: Both P and N remain unaltered, hence Invariant InvDualPN still holds after executing rule DecX.

Invariant InvDecs: The literal ℓ is a decision literal by definition. It is assigned decision level $d = \delta(I) + 1$. Since $\ell \in \mathsf{decs}(I\,\ell)$, we have $\delta(\mathsf{decs}(I\,\ell)) = \{1, \ldots, d\}$, and Invariant InvDecs holds after applying rule DecX.

Invariant InvImplIrred: Le n be arbitrary but fixed. Since ℓ is a decision literal, we have $P \wedge \mathsf{decs}_{\leqslant n}(I\,\ell) \equiv P \wedge \mathsf{decs}_{\leqslant n}(I) \wedge \ell \models I_{\leqslant n} \wedge \ell \equiv (I\,\ell)_{\leqslant n}$. Hence, Invariant InvImplIrred holds after applying rule DecX.

Invariant InvDSOP: The DSOP M remains unaltered by rule DecX, hence after applying rule DecX Invariant InvDSOP still holds .

DecYS.

The proofs of Invariants InvDualPN, InvDecs, InvDSOP, and InvDSOP are identical to the ones for rule DecX. □

6.3.2. Progress

Our method cannot get caught in an endless loop, as shown next.

Proposition 3. *EnumerateIrredundant always makes progress, i. e., in every non-terminal state a rule is applicable.*

Proof. The proof is executed by induction over the number of rule applications. We show that in any non-terminal state $(P,\ N,\ M,\ I,\ \delta)$ a rule is applicable.

Assume all variables are assigned and no conflict has occurred. If no relevant decision is left on the trail I, rule End1 can be applied. Otherwise, we execute an incremental SAT call $\mathsf{SAT}(N, \pi(I, X \cup Y))$. Since all input variables are assigned, we obtain a conflict by propagating internal variables only. Conflict analysis gives us the subsequence $I^\star$ of $\pi(I, X \cup Y)$ consisting of the literals involved in the conflict, which is a model of F. Since we are interested in the models of F projected onto X, we choose $B = \neg\mathsf{decs}(\pi(I^\star, X))$. Now, $\delta(B) = b+1$, and due to Invariant InvDecs, B contains exactly one decision literal ℓ such that $\delta(\ell) = b+1$ and therefore $\delta(B \setminus \{\ell\}) = b$. We choose J and K such that $I = J\,K$ and $b = \delta(J)$ and in particular $\ell|_K = 0$. After backtracking to decision level b, we have $I_{\leqslant b} = J$ where $B|_J = \{\ell\}$. All preconditions of rule Back1 are met.

If instead a conflict has occurred, there exists a clause $C \in P$ such that $C|_I = 0$. If $\delta(C) = 0$, rule End0 is applicable. Otherwise, by Invariant InvImplIrred we have $P \wedge \mathsf{decs}_{\leqslant\delta(I)}(I) \equiv P \wedge \mathsf{decs}_{\leqslant\delta(I)}(I) \wedge I_{\leqslant\delta(I)} \models I_{\leqslant\delta(I)}$. Since $I(P) \equiv 0$, also $P \wedge \mathsf{decs}_{\leqslant\delta(I)}(I) \wedge I_{\leqslant\delta(I)} \equiv P \wedge \mathsf{decs}_{\leqslant\delta(I)}(I) \equiv 0$. If we choose $D = \neg\mathsf{decs}(I)$ we obtain $P \wedge \neg D \wedge I_{\leqslant\delta(I)} \equiv 0$, thus $P \models D$. Clause D contains only decision literals and $\delta(D) = \delta(I)$. From Invariant InvDecs we know that D contains exactly one decision literal for each decision level in $\{1, \ldots, \delta(I)\}$. We choose $\ell \in D$ such that $\delta(\ell) = \delta(I)$. Then the asserting level is given by $j = \delta(D \setminus \{\ell\})$. Without loss of generalization we assume the trail to be of the form $I = J\,K$ where $\delta(J) = j$. After backtracking to decision level j, the trail is equal to J. Since $D|_J = \{\ell\}$, all conditions of rule Back0 hold.

If $P|_I \notin \{0, 1\}$, there are unassigned variables in $X \cup Y \cup S$. If there exists a clause $C \in P$ where $C|_I = \{\ell\}$, the preconditions of rule Unit are met. If instead $\mathsf{units}(F|_I) = \emptyset$, there exists a literal ℓ with $V(\ell) \in X \cup Y \cup S$ and $\delta(\ell) = \infty$. If not all relevant variables are assigned, the preconditions of rule DecX are satisfied. Otherwise, rule DecYS is applicable.

All possible cases are covered by this argument. Hence, in every non-terminal state a rule is applicable, i. e., EnumerateIrredundant always makes progress. □

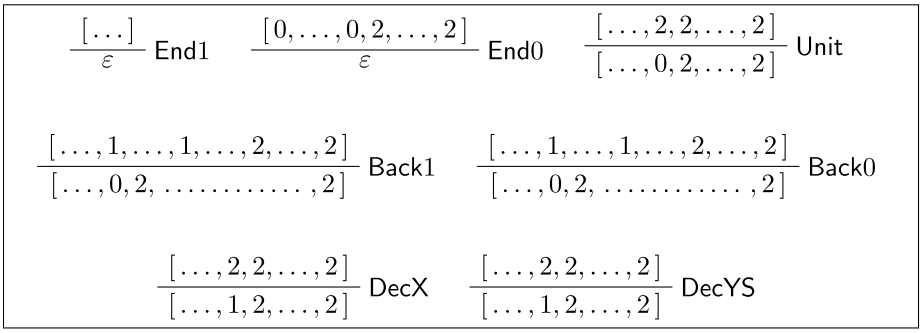


Fig. 8. Transitions of states mapped to lists according to Eq. (14). The initial state is depicted above the horizontal rule, the resulting state below. The two end rules lead to the minimal element ε representing the final state. Rule Unit replaces an unassigned variable (denoted by 2) by a propagated one (denoted by 0) and leaves the rest unchanged. Rules Back1 and Back0 replace a decision literal (denoted by 1) by a propagated one. Finally, the two decision rules replace an unassigned literal by a decision. Clearly, w.r.t. the lexicographic order, the states decrease by a rule application.

6.3.3. Termination

Proposition 4. *EnumerateIrredundant terminates.*

Proof. In our proof we follow the argument by Nieuwenhuis et al. [49] and Marić and Janičić [30], or more precisely the one by Blanchette et al. [8].

We need to show that from the initial state $(P,\ N,\ 0,\ \varepsilon,\ \delta_0)$ a final state M is reached in a finite number of steps, i.e., no infinite sequence of rule applications is generated. Otherwise stated, we need to prove that the relation $\rightsquigarrow_{\mathsf{EnumIrred}}$ is well-founded. To this end, we define a well-founded relation $\succ_{\mathsf{EnumIrred}}$ such that any transition $s \rightsquigarrow_{\mathsf{EnumIrred}} s'$ from a state s to a state s' implies $s \succ_{\mathsf{EnumIrred}} s'$.

In accordance with Blanchette et al. [8] but adopting the notation introduced by Fleury [18], we map states to lists. Using the abstract representation of the assignment trail I by Nieuwenhuis et al. [49], we write

$$I = I_0\, \ell_1\, I_1\, \ell_2\, I_2\, \ldots\, \ell_m\, I_m \quad \text{where} \quad \{\ell_1, \ldots, \ell_m\} = \mathsf{decs}(I). \tag{13}$$

The state $(P,\ N,\ M,\ I,\ \delta)$ is then mapped to

$$[\underbrace{0,\ldots,0}_{|I_0|},\ 1,\ \underbrace{0,\ldots,0}_{|I_1|},\ 1,\ \underbrace{0,\ldots,0}_{|I_2|},\ 1,\ \ldots,\ 1,\ \underbrace{0,\ldots,0}_{|I_m|},\ \underbrace{2,\ldots,2}_{|V|-|I|}] \tag{14}$$

where $V = X \cup Y \cup S$. In this representation, the order of the literals on I is reflected. Propagated literals are denoted by 0, decisions are denoted by 1. Unassigned variables are represented by 2 and are moved to the end. The final state M is represented by ε. The state containing the trail I in Eq. (13) is mapped to the list in Eq. (14). The first $|I_0|$ entries represent the literals propagated at decision level zero, the 1 at position $|I_0| + 1$ represents the decision literal ℓ_1, and so on for all decision levels on I. The last $|V| - |I|$ entries denote the unassigned variables. Notice that we are not interested in the variable assignment itself but in its structure, i.e., the number of propagated literals per decision level and the number of unassigned variables. Furthermore, the states are encoded into lists of the same length. This representation induces a lexicographic order $>_{\mathsf{lex}}$ on the states. We therefore define $\succ_{\mathsf{EnumIrred}}$ as the restriction of $>_{\mathsf{lex}}$ to $\{[\,v_1, \ldots, v_{|V|}] \mid v_i \in \{0, 1, 2\} \text{ for } 1 \leqslant i \leqslant |V|\}$. Accordingly, we have that $s \succ_{\mathsf{EnumIrred}} s'$, if $s >_{\mathsf{lex}} s'$.

In Fig. 8, the state transitions for the rules are visualized. In this representation, the unspecified elements occurring prior to the first digit are not altered by the application of the rule. We show that $s \succ_{\mathsf{EnumIrred}} s'$ for each rule, where s and s' encode the state before and after applying the corresponding rule, respectively, and end states are encoded by ε.

End1. The state s' is mapped to ε which is the minimal element with respect to $>_{\mathsf{lex}}$, hence $s \succ_{\mathsf{EnumIrred}} s'$ trivially holds. The representation of the state may contain both 0's and 1's but no 2's, since our algorithm detects only total models.[16] Recall that the associated trail must not contain any relevant decision, which is not reflected in the structure of the trail.

End0. The state s' is mapped to ε, which is the minimal element with respect to $>_{\mathsf{lex}}$, hence $s \succ_{\mathsf{EnumIrred}} s'$ trivially holds. The representation of the state may contain both 0's and 2's but no 1's, since any decision need be flipped.

Unit. An unassigned variable is propagated. Its representation changes from 2 to 0, and all elements preceding it remain unaffected. Due to $2 >_{\mathsf{lex}} 0$, we also have that $s \succ_{\mathsf{EnumIrred}} s'$.

Back1 / Back0. A decision literal, e.g., $\neg\ell$, is flipped and propagated at a lower decision level, let us say d. The decision level d is extended by ℓ, which is represented by 0 and replaces the decision literal at decision level $d+1$. All variables at

[16] This restriction may be weakened in favor of finding partial models. In this section, we refer to the rules introduced in Section 6.1 and discuss a generalization of our algorithm enabling the detection of partial models further down.

decision level $d+1$ and higher are unassigned and thus represented by 2. Therefore, $s \succ_{\mathsf{EnumIrred}} s'$. Notice that, although different preconditions of the rules Back1 and Back0 apply and the two rules differ, the structure of their states is the same.

DecX / DecYS. An unassigned variable is decided, i.e., the first occurrence of 2 is replaced by 1 in the representation. The other elements remain unaltered, hence $s \succ_{\mathsf{EnumIrred}} s'$. As for the backtracking rules, whether a relevant or irrelevant or internal variable is decided, is irrelevant and not reflected in the mapping of the state, as for rules Back1 and Back0.

We have shown that after any rule application the resulting state is smaller than the preceding one with respect to the lexicographic order on which $\succ_{\mathsf{EnumIrred}}$ is based. This argument shows that $\succ_{\mathsf{EnumIrred}}$ is well-founded and that therefore EnumerateIrredundant terminates. □

6.3.4. Equivalence

The final state is given by a DSOP M such that $M \equiv \pi(F, X)$. The proof is split into several steps. We start by proving that, given a total model I of P, its subsequence $I^\star$ returned by *SAT* (line 20 of EnumerateIrredundant in Fig. 3) is a (partial) model of $\pi(P, X)$ and that any total model of P found during execution either was already found or is found for the first time. Then we show that all models of P are found and that each model is found exactly once, before concluding by proving that $M \equiv \pi(F, X)$.

Proposition 5. *Let I be a total model of P and $I^\star = \mathsf{SAT}(N, \pi(I, X \cup Y))$. Then $I^\star$ is a model of $\pi(P, X)$.*

Proof. All variables in $X \cup Y \cup S$ are assigned and $P(X, Y, S)$ and $N(X, Y, T)$ are a dual representation of $F(X, Y)$. Invariant InvDualPN holds. In particular it holds for the values of the variables in $X \cup Y \cup S$ set to their values in I, i.e., we have that $\exists S\,[\,P(X, Y, S)|_I\,] \equiv \neg\exists T\,[\,N(X, Y, T)|_I\,]$ where only the unassigned variables in $(X \cup Y) - I$ are universally quantified. Since I is a total model of P, Invariant InvDualPN can be rewritten as $P(X, Y, S)|_I \equiv \neg\exists T\,[\,N(X, Y, T)|_I]$, and $\pi(I, X \cup Y)$ cannot be extended to a model of N. Since the variables in T are defined in terms of variables in $X \cup Y$, an incremental SAT call on $N \wedge I$ yields a conflict in N exclusively by propagating variables in T.

Exhaustive conflict analysis yields a clause D consisting of the negations of the (assumed) literals in I involved in the conflict. Its negation is a counter-model of $\pi(N, X \cup Y)$ which, due to Eq. (5), is a model of $\pi(P, X \cup Y)$. Obviously, the same holds for the projection onto X, and $\pi(\neg D, X) \models \pi(P, X)$. Since $I^\star = \pi(\neg D, X)$, we have $I^\star \models \pi(P, X)$, and the claim holds. □

Proposition 6. *A total model I of P is either*

(i) contained in M or
(ii) subsumed by a model in M or
(iii) a model of $P_0 \wedge \bigwedge_i B_i$ where B_i are the blocking clauses added to P_0

Proof. All variables in $X \cup Y \cup S$ are assigned and $I \models P$. If I was already found earlier, it was shrunken and the resulting model projected onto X to obtain $I^\star$ which was then added to M (rule Back1 and line 24 in EnumerateIrredundant in Fig. 3). If all assumed variables participated in the conflict and furthermore $Y = S = \emptyset$, then $\pi(I^\star, X \cup Y) = \pi(I^\star, X) = I$, and Item (i) holds. Otherwise, $I^\star < I$ and $I^\star$ subsumes I. Since $I^\star \in M$, in this case Item (ii) holds.

Suppose the model I is found for the first time. Since $I \models P$, also $I \models C$ for all clauses $C \in P$. This in particular holds for all blocking clauses which were added to the original formula $P = P_0$ (rule Back1 and line 22 in EnumerateIrredundant), and Item (iii) holds. □

Proposition 7. *Every model is found.*

Proof. According to Proposition 4, EnumerateIrredundant terminates. The final state M is reached by either rule End0 or rule End1. Assume $P = P_0 \bigwedge_i B_i$ with B_i denoting the blocking clauses added to the original formula P_0, and let I denote the current trail. If rule End0 is applied, then P is unsatisfiable, since $P|_I = 0$ and I contains no decision. If rule End1 is applied, then by Proposition 6, Item (iii), $m = \pi(I, X)$ is a model of P. Now, $(P \wedge \neg m)|_I = 0$ and I contains no decision literal. Therefore, $(P \wedge \neg m)|_I$ is unsatisfiable, and all models have been found. □

Proposition 8. *Every model is found exactly once.*

Proof. We recall Proposition 1 stating that only pairwise contradicting models are detected. In essence, this says that every model is found exactly once. □

Theorem 1 (*Correctness*)**.** *If $(P,\ N,\ 0,\ \varepsilon,\ \delta_0) \rightsquigarrow^*_{\mathsf{EnumIrred}} M$, then*

(i) $M \equiv \pi(F, X)$
(ii) $C_i \wedge C_j \equiv 0$ for $C_i, C_j \in M$ and $C_i \neq C_j$

Proof. The cubes in M are exactly the $I^\star$ computed from the total models of P. These are models of $\pi(P, X)$ (Proposition 5). Since by Proposition 7 all models are found, $M \equiv \pi(P, X)$. But by Eq. (2), $\mathsf{models}(\exists\, Y, S \,.\, P(X, Y, S)) = \mathsf{models}(\exists\, Y \,.\, F(X, Y))$ holds, i. e. $\mathsf{models}(\pi(P, X)) = \mathsf{models}(\pi(F, X))$. Therefore, $M \equiv \pi(F, X)$, and Item (i) holds.

Due to Proposition 1, the found models are pairwise contradicting, and Item (ii) holds as well. Notice that one could also use Proposition 8, since, as its consequence, only pairwise contradicting models are found. □

6.4. Generalization to partial model detection

EnumerateIrredundant only finds total models of P. In SAT solving, this makes sense from an computational point of view, because checking whether a partial assignment satisfies a formula is more expensive than extending it to a total one. However, model enumeration is computationally more expensive than SAT solving, hence satisfiability checks, e. g., in the form of entailment checks [41], might pay off. Notice that it still might make sense to shrink the models found. In this section, we discuss the changes to be made to our approach in order to support the detection of partial models.

First, the satisfiability condition need be changed such that it complies with any other strategy determining whether the (partial) assignment I satisfies P. The check now reads "$I(P) \equiv 1$" and replaces the one on line 15 of EnumerateIrredundant (Fig. 3). The rest of the algorithm remains unaltered. In our calculus (Fig. 5), the precondition "$I(P) \equiv 1$" replaces the check whether all variables in $X \cup Y \cup S$ are assigned in rules End1 and Back1. The other preconditions as well as rules End0, Unit, Back0, DecX, and DecYS remain unaltered.

Second, the computation of $I^\star$ need be adapted. It is based on the assumption that I is total such that a conflict in $N|_I$ is obtained by propagating only variables in T. Now Invariant InvDualPN ensures that a conflict in $N|_I$ is obtained also if I is a partial assignment, although in order to obtain this conflict variables in $X \cup Y$ might need be propagated or decided. Projecting the so-obtained assignment I' onto I solves the issue. Hence, we replace line 20 in EnumerateIrredundant (Fig. 3) by "$I' := \mathsf{SAT}(N, \pi(I, X \cup Y)); I^\star = \pi(I', V(I))$". These changes need also be reflected in rule Back1 and in the proof of Proposition 3.

7. Conflict-driven clause learning for redundant all-SAT

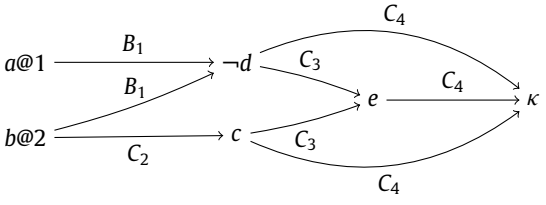
Conflict analysis is based on the assumption that the reason of every propagated literal is contained in the formula. In irredundant model enumeration (see Sections 5 and 6), this reason is either a clause learned by means of conflict-driven clause learning (CDCL) or a blocking clause. The motivation for adding blocking clauses is to ensure that the (partial) models detected by our calculus represent pairwise disjoint sets of total models. In some tasks, however, enumerating models multiple times causes no harm, and we can refrain from adding blocking clauses to the formula and avoid its blowing-up in size.

Consider a formula $P(X, Y, S)$ over relevant variables X, irrelevant variables Y and Tseitin variables S, and let I with variables in $X \cup Y \cup S$ be a total assignment satisfying P. Remember that by applying rule Back1, the trail I is shrunken to $I^\star$ and projected onto X obtaining $m = \pi(I^\star, X)$. Backtracking to decision level $\delta(m) - 1$ occurs, and the decision literal ℓ^d at decision level $\delta(m)$ is flipped, i. e., propagated with reason $B = \neg m$. If now B is not added to P and a conflict involving $\overline{\ell}$ occurs, the reason of $\overline{\ell}$ is not available for conflict analysis. To ensure the functioning of conflict analysis, we propose to annotate $\overline{\ell}$ on the trail with $\neg m$ but without adding B to P.

Example 10 (*Conflict Analysis for Model Enumeration*)**.** Consider the following formula over the set of variables $X = \{a, b, c, d\}$, $Y = \{e\}$, $S = \emptyset$:

$$P(X, Y, S) = \underbrace{(a \vee b \vee \neg c)}_{C_1} \wedge \underbrace{(\neg b \vee c)}_{C_2} \wedge \underbrace{(d \vee \neg c \vee e)}_{C_3} \wedge \underbrace{(d \vee \neg c \vee \neg e)}_{C_4}$$

Suppose we decide a and b, propagate c with reason C_2 and decide d followed by deciding e. The resulting trail $I_1 = a^d\, b^d\, c^{C_2}\, d^d\, e^d$ is a model of F. This model is blocked by $B_1 = (\neg a \vee \neg b \vee \neg d)$ with decision level $\delta(B_1) = 3$, which consists of the negated relevant decision literals on I_1. Considering only the decisions ensures that B_1 contains exactly one literal per decision level and that after backtracking to decision level $\delta(B_1) - 1 = 2$, the clause B_1 becomes unit. Recall that B_1 is not added to P. The decision literal d^d is flipped with reason B_1, and e is propagated with reason C_3. But now C_4 is falsified. The current trail is $I_2 = a^d\, b^d\, c^{C_2}\, \neg d^{B_1}\, e^{C_3}$, which is visualized by the following implication graph:



For conflict analysis, we first resolve the conflicting clause C_4 with the reason of e, C_3, obtaining the resolvent $(d \vee \neg c)$. Both d and $\neg c$ have the highest decision level 2, and we continue by resolving $(d \vee \neg c)$ with B_1 obtaining $(\neg c \vee \neg a \vee b)$, followed by resolution with C_2 resulting in $C_5 = (\neg a \vee \neg b)$, which has only one literal at decision level 2. The resolution process stops, and C_5 is added to P.

$$\textsf{Back1red}\colon\ (P,\, N,\, M,\, I,\, \delta) \rightsquigarrow_{\textsf{Back1red}} (P,\, N,\, M \vee m,\, J\ell^{B},\, \delta[K \mapsto \infty][\ell \mapsto b])$$
$$\text{if}\quad (X \cup Y \cup S) - I = \emptyset \quad \text{and} \quad \text{exists } I^{\star} \leqslant \pi(I, X \cup Y) \quad \text{with}$$
$$JK = I \quad \text{such that} \quad N \wedge I^{\star} \vdash_1 0 \quad \text{and} \quad m \stackrel{\text{def}}{=} \pi(I^{\star}, X) \quad \text{and}$$
$$B \stackrel{\text{def}}{=} \neg\textsf{decs}(m) \quad \text{and} \quad b + 1 \stackrel{\text{def}}{=} \delta(B) = \delta(m) \quad \text{and} \quad \ell \in B \quad \text{and}$$
$$\ell|_K = 0 \quad \text{and} \quad b = \delta(B \setminus \{\ell\}) = \delta(J)$$

Fig. 9. Rule for backtracking after detection of a model in redundant model enumeration. The calculus for redundant projected model enumeration differs from its irredundant counterpart only in the fact that no blocking clauses are used. Hence, all rules in Fig. 5 are maintained except for rule Back1, which is replaced by rule Back1red.

8. Projected redundant model enumeration

Now we turn our attention to the case where enumerating models multiple times is permitted. This allows for refraining from adding blocking clauses to the formula under consideration, since they might significantly slow down the enumerator. This affects both our algorithm and our calculus for irredundant projected model enumeration. Omitting the use of blocking clauses has a minor impact on our algorithm and its formalization. For this reason, in this section we point out the differences between the two methods.

8.1. Algorithm and calculus

The only difference compared to EnumerateIrredundant consists in the fact that no blocking clauses are added to P. However, they are remembered as annotations on the trail in order to enable conflict analysis after finding a model. Our algorithm EnumerateRedundant therefore is exactly the same as EnumerateIrredundant listed in Fig. 3 without lines 22–23. The annotation of flipped literals happens in function Backtrack() in line 26.

Accordingly, our formalization consists of all rules in Fig. 5 except for rule Back1 which is replaced by rule Back1red shown in Fig. 9. Rule Back1red differs from rule Back1 only in the fact that both P and N remain unaltered.

8.2. Example

Example 11 (*Projected Redundant Model Enumeration*)**.** Consider again Example 1 elaborated in detail in Section 6.2 for EnumerateIrredundant. We have

$$P = \underbrace{(a \vee c)}_{C_1} \wedge \underbrace{(a \vee \neg c)}_{C_2} \wedge \underbrace{(b \vee d)}_{C_3} \wedge \underbrace{(b \vee \neg d)}_{C_4}$$

and

$$\begin{aligned} N = {} & \underbrace{(\neg t_1 \vee \neg a)}_{D_1} \wedge \underbrace{(\neg t_1 \vee \neg c)}_{D_2} \wedge \underbrace{(a \vee c \vee t_1)}_{D_3} \wedge \\ & \underbrace{(\neg t_2 \vee \neg a)}_{D_4} \wedge \underbrace{(\neg t_2 \vee c)}_{D_5} \wedge \underbrace{(a \vee \neg c \vee t_2)}_{D_6} \wedge \\ & \underbrace{(\neg t_3 \vee \neg b)}_{D_7} \wedge \underbrace{(\neg t_3 \vee \neg d)}_{D_8} \wedge \underbrace{(b \vee d \vee t_3)}_{D_9} \wedge \\ & \underbrace{(\neg t_4 \vee \neg b)}_{D_{10}} \wedge \underbrace{(\neg t_4 \vee d)}_{D_{11}} \wedge \underbrace{(b \vee \neg d \vee t_4)}_{D_{12}} \wedge \\ & \underbrace{(t_1 \vee t_2 \vee t_3 \vee t_4)}_{D_{13}} \end{aligned}$$

Suppose $X = \{a, c\}$ and $Y = \{b, d\}$. The execution trail is depicted in Fig. 10.

Assume we decide a, b, c, and d (steps 1–4) obtaining the trail $I_1 = a^d\, b^d\, c^d\, d^d$ which is a model of P. Dual model shrinking occurs as in step 5 in the example elaborated in Section 6.2, except that the assumed literals b and c occur in a different order, and the same model $a\,b$ is obtained. Notice that the clause $B_1 = (\neg a \vee \neg b)$ is not added to P.

After backtracking, we have $P|_{I_1} = (d) \wedge (\neg d)$, and after propagating d (step 6), we obtain a conflict. The current trail is $I_3 = a^d\, \neg b^{B_1}\, d^{C_3}$ and $C_4|_{I_3} = ()$. Resolution of the reasons on I_3 in reverse assignment order is executed, starting with the conflicting clause C_4. We obtain $C_4 \otimes C_3 = (b) = C_5$, which contains exactly one literal at the maximum decision level, hence no further resolution steps are required. Since (b) is unit, the enumerator backtracks to decision level 0 and

Step	Rule	I	$P\|_I$	M
0		ε	P	0
1	DecX	a^d	$(b \vee d) \wedge (b \vee \neg d)$	0
2	DecX	$a^d\, b^d$	1	0
3	DecYS	$a^d\, b^d\, c^d$	1	0
4	DecYS	$a^d\, b^d\, c^d\, d^d$	1	0
5	Back1red	$a^d\, \neg b^{B_1}$	$(d) \wedge (\neg d)$	$a \wedge b$
6	Unit	$a^d\, \neg b^{B_1}\, d^{C_3}$	0	$a \wedge b$
7	Back0	b^{C_5}	$(a \vee c) \wedge (a \vee \neg c)$	$a \wedge b$
8	DecX	$b^{C_5}\, a^d$	1	$a \wedge b$
9	DecYS	$b^{C_5}\, a^d\, c^d$	1	$a \wedge b$
10	DecYS	$b^{C_5}\, a^d\, c^d\, d^d$	1	$a \wedge b$
11	Back1red	$b^{C_5}\, \neg a^{B_2}$	$(c) \wedge (\neg c)$	$(a \wedge b) \vee (b \wedge a)$
12	Unit	$b^{C_5}\, \neg a^{B_2}\, c^{C_1}$	0	$(a \wedge b) \vee (b \wedge a)$
13	End0			$(a \wedge b) \vee (b \wedge a)$

Fig. 10. Execution trace for $F = (a \vee c) \wedge (a \vee \neg c) \wedge (b \vee d) \wedge (b \vee \neg d)$ defined over the set of relevant variables $X = \{a, b\}$ and the set of irrelevant variables $Y = \{c, d\}$ (see Example 1).

InvDualPN: $\exists\, S\, [\, P(X, Y, S)\,] \equiv \neg \exists\, T\, [\, N(X, Y, T)\,]$

InvDecs: $\forall b \in \{1, \ldots, \delta(I)\}\, \exists! \ell \in \mathsf{decs}(I)\, [\delta(\ell) = b]$

InvImplRed: $\forall n \in \mathbb{N}\, .\, P \wedge \neg M \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$

Fig. 11. Invariants for projected model enumeration with repetition. Notice that Invariants InvDualPN and InvDecs are the same as for irredundant model enumeration while, due to the lack of blocking clauses, in Invariant InvImplIrred the models recorded in M need be considered.

propagates b with reason C_5 (step 7). After deciding a, c, and d, we find the same model $b\, a\, c\, d$ as in step 4 (steps 8–10). Obviously, model shrinking provides us with the same model $b\, a$, which is added to M, and the last relevant decision is flipped (step 11). Now unit propagation leads to a conflict (step 12), and since there are no decisions on the trail, the procedure stops (step 13). Now the cubes in M, which represent the models of P, are not pairwise disjoint anymore. However, we still have $M \equiv \pi(P, X) \equiv \pi(F, X)$.

8.3. Proofs

Invariants InvDualPN and InvDecs listed in Fig. 7 are applicable also for redundant model enumeration, since they involve none of P and N. Invariant InvImplIrred instead need be adapted since no blocking clauses are added to P and therefore it ceases to hold. Assume a model I has been found and shrunken to m and that the last relevant decision literal ℓ has been flipped. Since its reason $B = \mathsf{decs}(\neg m)$ is not added to P, from $P \wedge \mathsf{decs}(I)$ we cannot infer I. Recall that instead m is added to M, hence $\neg M$ contains the reasons of all decision literals which were flipped after having found a model. A closer look reveals that this case is analog to the one in our previous work [40]. In this work, we avoided the use of blocking clauses by means of chronological backtracking. However, the basic idea is the same, and we replace Invariant InvImplIrred by Invariant InvImplRed listed in Fig. 11. This is exactly Invariant (3) in our previous work on model counting [40], hence in our proof we use a similar argument. The invariants for redundant model enumeration under projection are given in Fig. 11.

8.3.1. Invariants in non-terminal states

Proposition 9 (*Invariants in EnumerateRedundant*)**.** *The Invariants InvDualPN, InvDecs, and InvImplRed hold in non-terminal states.*

Proof. The proof is carried out by induction over the number of rule applications. Assuming Invariants InvDualPN, InvDecs, and InvImplRed hold in a non-terminal state (P, N, M, I, δ), we show that they are met after the transition to another non-terminal state for all rules.

Now rules End1, End0, Back0, DecX, and DecYS are the same as for EnumerateIrredundant (Fig. 5). In Section 6.3.1 we already proved that after the execution of these rules Invariants InvDualPN and InvDecs still hold.

As for Invariant InvImplRed, from Proposition 6, Item (i) and Item (ii), and observing that $m \leqslant I$, where I is a total model of P and $m \in M$ its projection onto the relevant variables, we can conclude that Invariant InvImplRed holds as well. To see this, remember that in Invariant InvImplIrred we consider $P = P_0 \wedge \bigwedge_i B_i$ where the B_i are the clauses added to P_0 blocking the models m_i. But $B_i \leqslant m_i$, hence Invariant InvImplRed holds after applying rules Unit, Back0, DecX, and DecYS, and we are left to carry out the proof for rule Back1red.

Back1red

Invariant InvDualPN: Both P and N remain unaltered, therefore Invariant InvDualPN holds after the application of Back1red.

Invariant InvDecs: The proof is analogous to the one for rule Back1.

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

8.3.2. Progress and termination

The proofs that our method for redundant projected model enumeration always makes progress and eventually terminates are the same as in Sections 6.3.2 and 6.3.3.

8.3.3. Equivalence

Some properties proved for the case of irredundant model enumeration cease to hold if we allow enumerating redundant models. Specifically, Proposition 5, and Proposition 7 hold, while Proposition 8 does not. Item (i) and Item (ii) of Proposition 6 hold, while Item (iii) does not. In Theorem 1, Item (i) holds but Item (ii) does not. Their proofs remain the same as for irredundant model enumeration in Section 6.3.1.

8.4. Generalization

The same observations made for irredundant model enumeration in Section 6.4 apply.

9. Discussion

The complexity bounds for All-SAT are exponential in the number of variables occurring in the formula for all algorithms, due to the size of the search space. The goal is therefore to reduce the number of assignments to be checked, which is achieved by CDCL and adding short blocking clauses. In the brute-force approach for irredundant model enumeration, a blocking clause is exactly the negation of the satisfying assignment or consists of its negated decision literals [44,63]. In both cases, the resulting blocking clause has size at least the decision level of the trail, and their number corresponds to the number of models. The addition of blocking clauses to P (see line 22 of algorithm EnumerateIrredundant listed in Fig. 3) slows down unit propagation, in particular if they are long. Blocking many assignments with few clauses is therefore crucial, since CDCL-based SAT solvers spend most of their computing time with unit propagation.

For computing short blocking clauses, our dual model shrinking approach relies on the ability of conflict analysis to determine short clauses. Given a formula P and its negation N, it identifies the reason for the conflict in N induced by an assignment satisfying P by adopting CDCL in N. Due to the effectiveness of conflict analysis, our shrinking method has the potential to rule out a large number of assignments satisfying P as is shown by an example.

Example 12 (*Efficiency of Dual Model Shrinking*)**.** Consider the set of variables $X = \{a, b, c, d, e, f\}$ and the formula $F(X, Y) = (a \wedge e \wedge f) \vee (b \wedge e \wedge \neg f) \vee (c \wedge \neg e \wedge f) \vee (d \wedge \neg e \wedge \neg f)$. Without loss of generality, we assume $Y = \emptyset$. A CNF representation of F is given by

$$\begin{aligned} P(X, Y, S) = {} & (\neg v_1 \vee a) \wedge (\neg v_1 \vee e) \wedge (\neg v_1 \vee f) \wedge (v_1 \vee \neg a \vee \neg e \vee \neg f) \wedge \\ & (\neg v_2 \vee b) \wedge (\neg v_2 \vee e) \wedge (\neg v_2 \vee \neg f) \wedge (v_2 \vee \neg b \vee \neg e \vee f) \wedge \\ & (\neg v_3 \vee c) \wedge (\neg v_3 \vee \neg e) \wedge (\neg v_3 \vee f) \wedge (v_3 \vee \neg c \vee e \vee \neg f) \wedge \\ & (\neg v_4 \vee d) \wedge (\neg v_4 \vee \neg e) \wedge (\neg v_4 \vee \neg f) \wedge (v_4 \vee \neg d \vee e \vee f) \wedge \\ & (v_1 \vee v_2 \vee v_3 \vee v_4), \end{aligned}$$

where $S = \{v_1, v_2, v_3, v_4\}$. Let further a CNF representation $\neg F$ be given by

$$\begin{aligned} N(X, Y, T) = {} & (t_0) \wedge \\ & (\neg u_1 \vee \neg a \vee \neg e \vee \neg f) \wedge (u_1 \vee a) \wedge (u_1 \vee e) \wedge (u_1 \vee f) \wedge \\ & (\neg u_2 \vee \neg b \vee \neg e \vee f) \wedge (u_2 \vee b) \wedge (u_2 \vee e) \wedge (u_2 \vee \neg f) \wedge \\ & (\neg u_3 \vee \neg c \vee e \vee \neg f) \wedge (u_3 \vee c) \wedge (u_3 \vee \neg e) \wedge (u_3 \vee f) \wedge \\ & (\neg u_4 \vee \neg d \vee e \vee f) \wedge (u_4 \vee d) \wedge (u_4 \vee \neg e) \wedge (u_4 \vee \neg f) \wedge \\ & (\neg u_5 \vee u_1) \wedge (\neg u_5 \vee u_2) \wedge (\neg u_5 \vee u_3) \wedge (\neg u_5 \vee u_4) \wedge \\ & (u_5 \vee \neg u_1 \vee \neg u_2 \vee \neg u_3 \vee \neg u_4) \wedge \\ & (u_6 \vee t_0 \vee u_5) \wedge (u_6 \vee \neg t_0 \vee \neg u_5) \wedge \\ & (\neg u_6 \vee \neg t_0 \vee u_5) \wedge (\neg u_6 \vee t_0 \vee \neg u_5) \wedge \\ & (u_6) \end{aligned}$$

Suppose the model $I = a^d\, b^d\, c^d\, d^d\, e^d\, \neg v_3\, \neg v_4\, f^d\, \neg v_1\, \neg v_2$ of P has been found, where for better readability we omit the reasons of the propagation literals. Obviously, $I^\star = a\, e\, f$ already satisfies F, and its negation, the clause $\neg I^\star = (\neg a \vee \neg e \vee \neg f)$, is obtained after calling a SAT solver on N and $\pi(I, X)$ and analyzing the resulting conflict. Our algorithm for irredundant model enumeration enumerates exactly the cubes in F, which is optimal.

This conflict is obtained exclusively by unit propagation, which is linear in the length of I and the computation of $I^\star$ requires a linear number of resolution steps. Finally, the dual blocking clause encoding is linear in the size of the original formula. The enumerated models in Example 12 coincide with the ones obtained by our dual approach for projected model counting [38], which in our experiments finds minimal partial models, but without the overhead of processing two formulae throughout the computation. As already mentioned, for algorithm EnumerateIrredundant to be correct when adopting dual model shrinking in line 20, N need represent the negation of P anytime. Concretely, line 23 is mandatory in combination with dual model shrinking in line 20 for determining the backtracking level in line 25. Experiments further show that if a different, non-trivial blocking clause computation strategy is adopted in line 20, restarts are required, which might lead to repeating the same (partial) assignments multiple times [47,65].

The strength of executing entailment checks, as proposed as a generalization, is shown by Example 12. In fact, the trail $I = a^d\, b^d\, c^d\, d^d$ already logically entails P. The combination of logical entailment checks and dual model shrinking bears the potential to enumerate even shorter models. Recent experiments support our claim for the efficiency of executing logical entailment checks and of dual reasoning for model shrinking [19,61].

10. Conclusion

Model enumeration and projection, with and without repetition, is a key element to several tasks. We have presented two methods for propositional model enumeration under projection. EnumerateIrredundant uses blocking clauses to avoid enumerating models multiple times, while EnumerateRedundant is exempt from blocking clauses and admits repetitions. Our CDCL-based model enumerators detect total models and uses dual reasoning to shrink them.

To ensure correctness of the shrinking mechanism, we developed a dual encoding of the blocking clauses. We provided a formalization and proof of correctness of our blocking-based model enumeration approach and discussed a generalization to the case where partial models are found. These partial models might not be minimal, hence shrinking them still might make sense. Also, there is no guarantee that the shrunken models are minimal as they depend on the order of the variable assignments.

We presented a conflict-driven clause learning mechanism for redundant model enumeration, since standard CDCL might fail in the absence of blocking clauses. Basically, those clauses are remembered on the trail without being added to the input formula. This prevents a blowup of the formula but also does not further make use of these potentially short clauses, which in general propagate more eagerly than long clauses. We discussed the modifications of our blocking-based algorithm and calculus to support redundant model enumeration and provided a correctness proof. Intuitively, shorter partial models representing non-disjoint sets of total models might be found.

Our method does not guarantee that the shrunken model $I^\star$ is minimal w. r. t. the decision level b in line EnumerateIrredundant. However, finding short DSOPs is important in circuit design [37], and appropriate algorithms have been introduced by, e. g., Minato [36]. While DSOP minimization has been proven to be NP-complete [4], finding a smaller decision level b would already be advantageous, since besides restricting the search space to be explored it generates shorter models. To this end, we plan to adapt our dual shrinking algorithm to exploit the Tseitin encoding as proposed by Iser et al. [25].

In the presence of multiple conflicting clauses, a related interesting question might also be which one to choose as a starting point for conflict analysis with the aim to backtrack as far as possible. This is not obvious unless all conflicts are analyzed.

Determining short models makes our approach suitable for circuit design. We are convinced that this work provides incentives not only for the hardware-near community but also for the enumeration community.

Acknowledgments

The work was supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria, by the QuaSI project funded by D-Wave Systems Inc., and by the Italian Association for Artificial Intelligence (AI*IA). We acknowledge the support of the MUR PNRR project FAIR – Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU. The work was partially supported by the project "AI@TN" funded by the Autonomous Province of Trento. This research was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation program under GA No 952215. We also thank Mathias Fleury for a number of helpful discussions which in particular led to a more elegant equivalence proof compared to our original version. We are also indebted to the anonymous reviewers for their valuable feedback.

Data availability

No data was used for the research described in the article.

References

[1] R.A. Aziz, G. Chu, C.J. Muise, P.J. Stuckey, #∃SAT: Projected model counting, in: SAT, in: Lecture Notes in Computer Science, vol. 9340, Springer, 2015, pp. 121–137.
[2] F. Bacchus, S. Dalmao, T. Pitassi, DPLL with caching: A new algorithm for #SAT and Bayesian inference, Electron. Colloquium Comput. Complex. 10 (003) (2003).
[3] R.J. Bayardo Jr., J.D. Pehoushek, Counting models using connected components, in: AAAI/IAAI, AAAI Press / The MIT Press, 2000, pp. 157–162.
[4] A. Bernasconi, V. Ciriani, F. Luccio, L. Pagli, Compact DSOP and partial DSOP forms, Theory Comput. Syst. 53 (4) (2013) 583–608.
[5] A. Biere, A. Cimatti, E.M. Clarke, Y. Zhu, Symbolic model checking without BDDs, in: TACAS, in: Lecture Notes in Computer Science, vol. 1579, Springer, 1999, pp. 193–207.
[6] A. Biere, S. Hölldobler, S. Möhle, An abstract dual propositional model counter, in: YSIP, in: CEUR Workshop Proceedings, vol. 1837, CEUR-WS.org, 2017, pp. 17–26.
[7] E. Birnbaum, E.L. Lozinskii, The good old Davis-Putnam procedure helps counting models, J. Artif. Intell. Res. 10 (1999) 457–477.
[8] J.C. Blanchette, M. Fleury, P. Lammich, C. Weidenbach, A verified SAT solver framework with learn, forget, restart, and incrementality, J. Autom. Reason. 61 (1–4) (2018) 333–365.
[9] J. Brauer, A. King, J. Kriener, Existential quantification as incremental SAT, in: CAV, in: LNCS, vol. 6806, Springer, 2011, pp. 191–207.
[10] M. Cadoli, F.M. Donini, A survey on knowledge compilation, AI Commun. 10 (3-4) (1997) 137–150.
[11] M. Chavira, A. Darwiche, On probabilistic inference by weighted model counting, Artif. Intell. 172 (6-7) (2008) 772–799.
[12] A. Darwiche, P. Marquis, A knowledge compilation map, J. Artif. Intell. Res. 17 (2002) 229–264.
[13] M. Davis, G. Logemann, D.W. Loveland, A machine program for theorem-proving, Commun. ACM 5 (7) (1962) 394–397.
[14] M. Davis, H. Putnam, A computing procedure for quantification theory, J. ACM 7 (3) (1960) 201–215.
[15] J.M. Dudek, V.H.N. Phan, M.Y. Vardi, DPMC: weighted model counting by dynamic programming on project-join trees, in: CP, in: Lecture Notes in Computer Science, vol. 12333, Springer, 2020, pp. 211–230.
[16] N. Eén, N. Sörensson, Temporal induction by incremental SAT solving, Electron. Notes Theor. Comput. Sci. 89 (4) (2003) 543–560.
[17] J.K. Fichte, M. Hecher, S. Woltran, M. Zisser, Weighted model counting on the GPU by exploiting small treewidth, in: ESA, in: LIPIcs, vol. 112, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 28:1–28:16.
[18] M. Fleury, Formalisation of Ground Inference Systems in a Proof Assistant (Master's thesis), École normale supérieure de Rennes, 2015, URL https://www.mpi-inf.mpg.de/fileadmin/inf/rg1/Documents/fleury_master_thesis.pdf.
[19] D. Fried, A. Nadel, R. Sebastiani, Y. Shalmon, Entailing generalization boosts enumeration, in: SAT, in: LIPIcs, vol. 305, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, pp. 13:1–13:14.
[20] M. Gebser, B. Kaufmann, T. Schaub, Solution enumeration for projected boolean search problems, in: CPAIOR, in: Lecture Notes in Computer Science, vol. 5547, Springer, 2009, pp. 71–86.
[21] O. Grumberg, A. Schuster, A. Yadgar, Memory efficient all-solutions SAT solver and its application for reachability analysis, in: FMCAD, in: LNCS, vol. 3312, Springer, 2004, pp. 275–289.
[22] A. Gupta, Z. Yang, P. Ashar, A. Gupta, SAT-based image computation with application in reachability analysis, in: FMCAD, in: LNCS, vol. 1954, Springer, 2000, pp. 354–371.
[23] J.N. Hooker, Solving the incremental satisfiability problem, J. Log. Program. 15 (1&2) (1993) 177–186.
[24] J. Huang, A. Darwiche, The language of search, J. Artif. Intell. Res. 29 (2007) 191–219.
[25] M. Iser, C. Sinz, M. Taghdiri, Minimizing models for Tseitin-encoded SAT instances, in: SAT, in: Lecture Notes in Computer Science, vol. 7962, Springer, 2013, pp. 224–232.
[26] H. Jin, H. Han, F. Somenzi, Efficient conflict analysis for finding all satisfying assignments of a boolean circuit, in: TACAS, in: Lecture Notes in Computer Science, vol. 3440, Springer, 2005, pp. 287–300.
[27] J. Lagniez, P. Marquis, An improved decision-DNNF compiler, in: IJCAI, ijcai.org, 2017, pp. 667–673.
[28] S.K. Lahiri, R. Nieuwenhuis, A. Oliveras, SMT techniques for fast predicate abstraction, in: CAV, in: LNCS, vol. 4144, Springer, 2006, pp. 424–437.
[29] B. Li, M.S. Hsiao, S. Sheng, A novel SAT all-solutions solver for efficient preimage computation, in: DATE, IEEE Computer Society, 2004, pp. 272–279.
[30] F. Marić, P. Janičić, Formalization of abstract state transition systems for SAT, Log. Methods Comput. Sci. 7 (3) (2011).
[31] J.P. Marques-Silva, K.A. Sakallah, GRASP: A search algorithm for propositional satisfiability, IEEE Trans. Comput. 48 (5) (1999) 506–521.
[32] K.L. McMillan, Applying SAT methods in unbounded symbolic model checking, in: CAV, in: LNCS, vol. 2404, Springer, 2002, pp. 250–264.
[33] K.L. McMillan, Interpolation and SAT-based model checking, in: CAV, in: Lecture Notes in Computer Science, vol. 2725, Springer, 2003, pp. 1–13.
[34] P.B. Miltersen, J. Radhakrishnan, I. Wegener, On converting CNF to DNF, in: MFCS, in: Lecture Notes in Computer Science, vol. 2747, Springer, 2003, pp. 612–621.
[35] P.B. Miltersen, J. Radhakrishnan, I. Wegener, On converting CNF to DNF, Theor. Comput. Sci. 347 (1-2) (2005) 325–335.
[36] S. Minato, Fast generation of prime-irredundant covers from binary decision diagrams, IEICE Trans. Fundam. E76-A (6) (1993) 967–973.

[37] S. Minato, G. De Micheli, Finding all simple disjunctive decompositions using irredundant sum-of-products forms, in: ICCAD, ACM / IEEE Computer Society, 1998, pp. 111–117.
[38] S. Möhle, A. Biere, Dualizing projected model counting, in: ICTAI, IEEE, 2018, pp. 702–709.
[39] S. Möhle, A. Biere, Backing backtracking, in: SAT, in: Lecture Notes in Computer Science, vol. 11628, Springer, 2019a, pp. 250–266.
[40] S. Möhle, A. Biere, Combining conflict-driven clause learning and chronological backtracking for propositional model counting, in: GCAI, in: EPiC Series in Computing, vol. 65, EasyChair, 2019b, pp. 113–126.
[41] S. Möhle, R. Sebastiani, A. Biere, Four flavors of entailment, in: SAT, in: Lecture Notes in Computer Science, vol. 12178, Springer, 2020, pp. 62–71.
[42] P. Morettin, A. Passerini, R. Sebastiani, Efficient weighted model integration via SMT-based predicate abstraction, in: IJCAI, ijcai.org, 2017, pp. 720–728.
[43] P. Morettin, A. Passerini, R. Sebastiani, Advanced SMT techniques for weighted model integration, Artif. Intell. 275 (2019) 1–27.
[44] A. Morgado, J.P.M. Silva, Good learning and implicit model enumeration, in: ICTAI, IEEE Computer Society, 2005, pp. 131–136.
[45] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an efficient SAT solver, in: DAC, ACM, 2001, pp. 530–535.
[46] C.J. Muise, S.A. McIlraith, J.C. Beck, E.I. Hsu, Dsharp: Fast d-DNNF compilation with sharpSAT, in: Canadian Conference on AI, in: Lecture Notes in Computer Science, vol. 7310, Springer, 2012, pp. 356–361.
[47] A. Nadel, V. Ryvchin, Chronological backtracking, in: SAT, in: LNCS, vol. 10929, Springer, 2018, pp. 111–121.
[48] A. Niemetz, M. Preiner, A. Biere, Turbo-charging Lemmas on demand with don't care reasoning, in: FMCAD, IEEE, 2014, pp. 179–186.
[49] R. Nieuwenhuis, A. Oliveras, C. Tinelli, Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T), J. ACM 53 (6) (2006) 937–977.
[50] H. Palacios, B. Bonet, A. Darwiche, H. Geffner, Pruning conformant plans by counting models on compiled d-DNNF representations, in: ICAPS, AAAI, 2005, pp. 141–150.
[51] D.A. Plaisted, S. Greenbaum, A structure-preserving clause form translation, J. Symb. Comput. 2 (3) (1986) 293–304.
[52] K. Ravi, F. Somenzi, Minimal assignments for bounded model checking, in: TACAS, in: LNCS, vol. 2988, Springer, 2004, pp. 31–45.
[53] J.A. Robinson, A machine-oriented logic based on the resolution principle, J. ACM 12 (1) (1965) 23–41.
[54] T. Sang, P. Beame, H.A. Kautz, Performing Bayesian inference by weighted model counting, in: AAAI, AAAI Press / The MIT Press, 2005, pp. 475–482.
[55] R. Sebastiani, Lazy satisfiability modulo theories, J. Satisf. Boolean Model. Comput. 3 (3-4) (2007) 141–224.
[56] R. Sebastiani, Are you satisfied by this partial assignment?, 2020, https://arxiv.org/abs/2003.04225.
[57] S. Sheng, M.S. Hsiao, Efficient preimage computation using a novel success-driven ATPG, in: DATE, IEEE Computer Society, 2003, pp. 10822–10827.
[58] O. Shtrichman, Tuning SAT checkers for bounded model checking, in: CAV, in: LNCS, vol. 1855, Springer, 2000, pp. 480–494.
[59] O. Shtrichman, Pruning techniques for the SAT-based bounded model checking problem, in: CHARME, in: LNCS, vol. 2144, Springer, 2001, pp. 58–70.
[60] J.P.M. Silva, K.A. Sakallah, GRASP - a new search algorithm for satisfiability, in: ICCAD, IEEE Computer Society / ACM, 1996, pp. 220–227.
[61] G. Spallitta, R. Sebastiani, A. Biere, Disjoint partial enumeration without blocking clauses, in: AAAI, AAAI Press, 2024, pp. 8126–8135.
[62] A. Sullivan, D. Marinov, S. Khurshid, Solution enumeration abstraction: A modeling idiom to enhance a lightweight formal method, in: ICFEM, in: Lecture Notes in Computer Science, vol. 11852, Springer, 2019, pp. 336–352.
[63] T. Toda, T. Soh, Implementing efficient all solutions SAT solvers, ACM J. Exp. Algorithmics 21 (1) (2016) 1.12:1–1.12:44.
[64] G. Tseitin, On the complexity of derivation in propositional calculus, Stud. Constr. Math. Math. Log. (1968) 115–125.
[65] P. van der Tak, A. Ramos, M. Heule, Reusing the assignment trail in CDCL solvers, J. Satisf. Boolean Model. Comput. 7 (4) (2011) 133–138.
[66] I. Wegener, The Complexity of Boolean Functions, Wiley-Teubner, 1987.
[67] C. Zengler, W. Küchlin, Boolean quantifier elimination for automotive configuration - A case study, in: FMICS, in: Lecture Notes in Computer Science, vol. 8187, Springer, 2013, pp. 48–62.