Four Flavors of Entailment

Sibylle Möhle[1], Roberto Sebastiani[2], and Armin Biere[1]

[1] Johannes Kepler University Linz, Austria
[2] DISI, University of Trento, Italy

Abstract. We present a novel approach for enumerating partial models of a propositional formula, inspired by how theory solvers and the SAT solver interact in lazy SMT. Using various forms of dual reasoning allows our CDCL-based algorithm to enumerate partial models with no need for exploring and shrinking full models. Our focus is on model enumeration without repetition, with potential applications in weighted model counting and weighted model integration for probabilistic inference over Boolean and hybrid domains. Chronological backtracking renders the use of blocking clauses obsolete. We provide a formalization and examples. We further discuss important design choices for a future implementation related to the strength of dual reasoning, including unit propagation, using SAT or QBF oracles.

1 Introduction

Model enumeration is a key task in various activities, such as lazy Satisfiability Modulo Theories [29], predicate abstraction [13], software product line engineering [7], model checking [2,18,31], and preimage computation [14,30].

Whereas in some applications enumerating models multiple times causes no harm, in others avoiding repetitions is crucial. Examples are weighted model counting (WMC) for probabilistic reasoning in Boolean domains and weighted model integration (WMI), which generalizes WMC for hybrid domains [22,23]. There, the addends are *partial* satisfying assignments, i.e., some variables remain unassigned. Each of these assignments represents a set of *total* assignments, and consequently, the number of the addends is reduced. A formula might be represented in a concise manner by the disjunction of its pairwise contradicting partial models, which is of interest in digital circuit synthesis [1]. Partial models are relevant also in predicate abstraction [13], preimage computation [14,30], and existential quantification [4]. They can be obtained by shrinking total models [32]. Alternatively, dual reasoning, where the formula is considered together with its negation, allows for pruning the search space early and detecting partial models. It is also applied in the context of model counting [3,19].

If only a subset X of the variables is significant, the models are *projected* onto these *relevant* variables. We say that we *existentially quantify* the formula over the *irrelevant* variables Y and write $\exists Y\,[\,F(X,Y)\,]$, where $F(X,Y)$ is a formula over variables X and Y such that $X \cap Y = \emptyset$. Projected model enumeration occurs in automotive configuration [34], existential quantifier elimination [4], image computation [9,10], predicate abstraction [13], and bounded model checking [31].

To avoid finding models multiple times, blocking clauses might be added to the formula under consideration [11,18]. This method suffers from a potentially exponential blowup of the formula and consequent slowdown of unit propagation. Toda and Soh [33] address this issue by a variant of conflict analysis, which is motivated by Gebser et al. [8] and is exempt from blocking clauses. Chronological backtracking in Grumberg et al. [9] and our previous work [21] ensures that the search space is traversed in a systematic manner, similarly to DPLL [5], and the use of blocking clauses is avoided. Whenever a model is found, the last (relevant) decision literal is flipped. No clause asserting this flipped decision is added, which might cause problems during later conflict analysis. This problem is addressed by modifying the implication graph [9] or by an alternative first UIP scheme [33].

Our contribution. We lift the way how theory and SAT solver interact in SMT to propositional projected model enumeration without repetition. Based on the notion of logical entailment, combined with dual reasoning, our algorithm detects partial models in a forward manner, rendering model shrinking superfluous. The test for entailment is crucial in our algorithm. Anticipating a future implementation, we present it in four flavors with different strengths together with examples. The main enumeration engine uses chronological CDCL [25], is exempt from blocking clauses, and thus does not suffer from a formula blowup. Its projection capabilities make it suitable also for applications requiring model enumeration with projection. We conclude our presentation by a formalization of our algorithm and a discussion of the presented approach. Our work is motivated by projected model counting and weighted model integration. We therefore focus on (projected) model enumeration without repetition. Contrarily to Oztok and Darwiche [26], we use an oracle and build a Disjoin Sum-of-Products (DSOP) [1]. The work by Lagniez and Marquis [12] is orthogonal to ours. It is led by a disjunctive decomposition of the formula under consideration after a full model is found and also decomposes it into disjoint connected components.

2 Preliminaries

A *literal* ℓ is a variable v or its negation $\neg v$. We denote by $V(\ell)$ the variable of ℓ and extend this notation to sets and sequences of literals. We write $\bar{\ell}$ for the complement of ℓ, i.e., $\bar{\ell} = \neg\ell$, defining $\neg\neg\ell = \ell$. A formula in *conjunctive normal form (CNF)* over variables V is defined as a conjunction of *clauses*, which are disjunctions of literals with variable in V, whereas a formula in *disjunctive normal form (DNF)* is a disjunction of *cubes*, which are conjunctions of literals. We might interpret formulae, clauses, and cubes also as sets of clauses or cubes, and literals and write $C \in F$ for referring to a clause or cube C in a formula F and $\ell \in C$ where ℓ is a literal in C. The empty CNF formula and the empty cube are denoted by 1, the empty DNF formula and the empty clause by 0.

A *total assignment* is a mapping from the set of variables V to the truth values 1 (true) and 0 (false). A *trail* $I = \ell_1 \dots \ell_n$ is a non-contradictory sequence of literals, which might also be interpreted as a *(possibly partial) assignment*, where $I(\ell) = 1$ if $\ell \in I$ and $I(\ell) = 0$ if $\neg\ell \in I$. We denote the empty trail by ε

and the set of variables of the literals on I by $V(I)$. Trails and literals might be concatenated, written $I = JK$ and $I = J\ell$, provided $V(J) \cap V(K) = \emptyset$ and $V(J) \cap V(\ell) = \emptyset$. We interpret I also as a set of literals and write $\ell \in I$ to denote a literal ℓ on I. The *residual* of a formula F under a trail I, written $F|_I$, is obtained by replacing the literals ℓ in F, where $V(\ell) \in V(I)$, by their truth value, and by recursively propagating truth values through Boolean connectives. In particular, for a CNF formula this consists in removing satisfied clauses as well as falsified literals. By "=" in $F|_I = 1$ and $F|_I = 0$, notably by omitting quantifiers, we explicitly mean syntactical equality and consider the (possibly partial) assignment represented by I, i.e., only the literals on I. The notion of residual is extended similarly to clauses and literals. We denote by $X - I$ the unassigned variables in X. By $\pi(I, X)$ we refer to the projection of I onto X and extend this notation to sets of literals.

The *decision level function* $\delta \colon V \mapsto \mathbb{N} \cup \{\infty\}$ returns the decision level of a variable v. If v is unassigned, we have $\delta(v) = \infty$, and δ is updated whenever v is assigned or unassigned. We define $\delta(\ell) = \delta(V(\ell))$ for a literal ℓ, $\delta(C) = \mathsf{max}\{\delta(\ell) \mid \ell \in C\}$ for a clause $C \neq 0$, and $\delta(I) = \mathsf{max}\{\delta(\ell) \mid \ell \in I\}$ for a sequence of literals $I \neq \varepsilon$. Further, $\delta(L) = \mathsf{max}\{\delta(\ell) \mid \ell \in L\}$ for a set of literals $L \neq \emptyset$. We define $\delta(0) = \delta(\varepsilon) = \delta(\emptyset) = 0$. The updated function δ, in which $V(\ell)$ is assigned to decision level d, is denoted by $\delta[\ell \mapsto d]$. If all literals in V are unassigned, we write $\delta[V \mapsto \infty]$ or $\delta \equiv \infty$. The function δ is left-associative, i.e., $\delta[I \mapsto \infty][\ell \mapsto d]$ first unassigns all literals on I and then assigns literal ℓ to decision level d. We mark the decision literals on I by a superscript, i.e., ℓ^d, and denote the set consisting of the decision literals on I by $\mathsf{decs}(I) = \{\ell \mid \ell^d \in I\}$. Similarly, we denote the set of unit literals in F or its residual under I by $\mathsf{units}(F)$ or $\mathsf{units}(F|_I)$. Trails are partitioned into *decision levels*, and $I_{\leqslant n}$ is the subsequence of I consisting of all literals ℓ where $\delta(\ell) \leqslant n$.

Following Sebastiani [28], we say that a (partial) assignment I *entails* a formula F, if all total extensions of I satisfy F. In this work it was noticed that, if I entails F, we can not conclude that $F|_I = 1$, but only that $F|_I$ is valid. Consider as an example $F = (x \wedge y) \vee (x \wedge \neg y)$ over variables $X = \{x\}$ and $Y = \{y\}$ and the trail $I = x$ ranging over $X \cup Y$. The possible extensions of I are $I' = xy$ and $I'' = x\neg y$. We have $F|_{I'} = F|_{I''} = 1$, therefore I entails F. Notice that $F|_I = y \vee \neg y$ is valid but it syntactically differs from 1.

3 Early Pruning for Projected Model Enumeration

Our approach is inspired by how theory solvers and the SAT solver interact in lazy SMT. A general schema is described in Fig. 1. Let $F(X, Y)$ be a formula over relevant variables X and irrelevant variables Y such that $X \cap Y = \emptyset$. A SAT solver executes enumeration, either DPLL-based [5,6] or CDCL-based [17,24], on F, maintaining a trail I over variables $X \cup Y$. In lines 1–16 and 23–24, we consider the CDCL-based enumeration engine with chronological backtracking of our framework [21]. Now assume unit propagation has been carried out until completion, no conflict occurred and there are still unassigned variables (line 17).

Input: formula $F(X, Y)$ over variables $X \cup Y$ such that $X \cap Y = \emptyset$, trail I, decision level function δ

Output: DNF M consisting of models of F projected onto X

```
Enumerate ( F )
1  I := ε                                          // empty trail
2  δ := ∞                                          // unassign all variables
3  M := 0                                          // empty DNF
4  forever do
5      C := PropagateUnits ( F, I, δ )
6      if C ≠ 0 then                               // conflict
7          c := δ(C)                               // conflict level
8          if c = 0 then
9              return M
10         AnalyzeConflict ( F, I, C, c )
11     else if all variables in X ∪ Y are assigned then   // I is total model
12         if V(decs(I)) ∩ X = ∅ then              // no relevant decision left
13             return M ∨ π(I, X)                  // record I projected onto X
14         M := M ∨ π(I, X)
15         b := δ(decs(π(I, X)))                   // highest relevant decision level
16         Backtrack ( I, b − 1 )                  // flip last relevant decision
17     else if Entails ( I, F ) then               // I is partial model
18         if V(decs(I)) ∩ X = ∅ then              // no relevant decision left
19             return M ∨ π(I, X)                  // record I projected onto X
20         M := M ∨ π(I, X)
21         b := δ(decs(π(I, X)))                   // highest relevant decision level
22         Backtrack ( I, b − 1 )                  // flip last relevant decision
23     else
24         Decide ( I, δ )
```

Fig. 1. Early pruning for projected model enumeration. Lines 1–16 and 23–24 list CDCL-based model enumeration with chronological backtracking. If after unit propagation no conflict occurs and not all variables are assigned, an oracle might be called to check whether I entails F (line 17). If Entails returns 1, the relevant decision literal with highest decision level might be flipped. Otherwise, a decision is taken (line 24). Notice that lines 12–16 and lines 18–22 are identical.

The trail I already might entail F, although $F|_I \neq 1$. We can check whether I entails F by an incremental call to an "oracle" [16] Entails on I and F. If Entails returns 1, then the procedure does not need to test any total extension of I, since all of them are models of F. It can proceed and flip the relevant decision literal with highest decision level (line 21–22). If Entails returns 0, a decision needs to be taken (line 24). Notice that lines 12–16 and lines 18–22 are identical. Our method is based on chronological backtracking and follows the scheme in our framework [21], the functions PropagateUnits() and AnalyzeConflict() are taken from our previous work [20]. Entails plays the role of an "early pruning call" to a

theory solver in SMT, and F plays the role of the theory [29]. Redundant work is saved by applying unit propagation until completion before calling Entails.

Quantified entailment condition. We use quantifiers with QBF semantics, and quantified formulae are always closed. A closed QBF formula evaluates to either 1 or 0. Consider $\varphi = \forall X \forall Y \, [\, F|_I \,]$, where F is a formula over variables $X \cup Y$ and the trail I ranges over $X \cup Y$. In φ, the remaining variables $(X \cup Y) - I$ are quantified. Accordingly, by $\forall X \forall Y \, [\, F|_I \,] = 1$, we express that all possible total extensions of I satisfy F, in contrast to $F|_I = 1$, expressing syntactic equality according to Sect. 2. The latter fact implies the former, but not vice versa.

Entailment under projection. If Entails implements the notion of entailment described in Sect. 2, then by calling it on I and F, we check whether $F|_J = 1$ for all total extensions J of I, i.e., whether $\forall X \forall Y \, [\, F|_I \,] = 1$. However, since we are interested in the models of F projected onto X, it suffices to check that for each possible assignment J_X to the unassigned variables in X, there exists *one* assignment J_Y to the unassigned variables in Y such that $F|_{I'} = 1$ where $I' = I \cup J_X \cup J_Y$. In essence, we need to determine the truth of the QBF formula $\forall X \exists Y \, [\, F|_I \,]$, which, in general, might be expensive, computationally. In some cases, however, a computationally cheaper (but weaker) test might be sufficient. Entails in line 17 of Enumerate can be seen as a black box pooling four entailment tests of different strengths, which we discuss in the next section.

4 Testing Entailment

Consider the original entailment condition, $\forall X \forall Y \, [\, F|_I \,] = 1$. Now we have that $\forall X \forall Y \, [\, F|_I \,] = 1 \iff \exists X \exists Y \, [\, \neg F|_I \,] = 0$. Therefore, to check whether I entails F, a SAT solver might be called to check whether $\neg F \wedge I$ is unsatisfiable. The SAT solver returns "unsat", if and only if I entails F. This observation motivates the use of dual reasoning for testing entailment in cases where cheaper tests fail. We present four flavors of the entailment test and provide examples.

1) $F|_I = 1$ (*syntactic check*). If $F|_I = 1$, also $\forall X \forall Y \, [\, F|_I \,] = 1$, and I entails F.
2) $F|_I \approx 1$ (*incomplete check in* **P**). Alternatively, if $F|_I$ differs from 1, an incomplete algorithm might be used, to check whether $\neg F \wedge I$ is unsatisfiable, by for instance executing only unit propagation or aborting after a predefined number of decision levels.
3) $F|_I \equiv 1$ (*semantic check in* **coNP**). A SAT oracle runs on $\neg F \wedge I$ until termination. Basically, it checks the unsatisfiability of $\neg F \wedge I$, i.e., whether it holds that $\exists X \exists Y \, [\, \neg F|_I \,] = 0$. If it answers "unsat", then I entails F.
4) $\forall X \exists Y \, [F|_I \,] = 1$ (*check in* $\mathbf{\Pi}_2^P$). A QBF oracle is called to check whether the 2QBF formula $\forall X \exists Y \, [F|_I \,]$ is 1.

Modern SAT solvers mostly work on CNFs. Thus, following our dualization approach [19], we may convert $F(X, Y)$ and $\neg F(X, Y)$ into CNF formulae $P(X, Y, S)$ and $N(X, Y, T)$, where S and T denote the variables introduced by the CNF encoding. Notice that $I \wedge \neg F$ is unsatisfiable iff $I \wedge N$ is unsatisfiable.

Table 1. Examples of formulae F over relevant variables X and irrelevant variables Y. For a concise representation of formulae, we represent conjunction by juxtaposition and negation by overline. In all examples, I entails F projected onto X. The entailment tests are listed from left to right in ascending order by their strength. Here, "✓" denotes the fact that I passes the test in the column, if applied to the formula in the row.

F	X	Y	I	$=1$	≈ 1	$\equiv 1$	2QBF
$(x_1 \vee y \vee x_2)$	$\{x_1, x_2\}$	$\{y\}$	x_1	✓	✓	✓	✓
$x_1 y \vee \overline{y} x_2$	$\{x_1, x_2\}$	$\{y\}$	$x_1 x_2$		✓	✓	✓
$x_1(\overline{x_2}\,\overline{y} \vee \overline{x_2} y \vee x_2 \overline{y} \vee x_2 y)$	$\{x_1, x_2\}$	$\{y\}$	x_1			✓	✓
$x_1(x_2 \leftrightarrow y)$	$\{x_1, x_2\}$	$\{y\}$	x_1				✓

Table 1 lists four examples, which differ in the strength of the required entailment test. The first column lists the formula F, the second and third column show the definitions of X and Y. For a concise representation of formulae, we represent conjunction by juxtaposition and negation by overline. The fourth column contains the current trail I. The fifth to eighth column denote the tests, in ascending order by their strength: $F|_I = 1$, $F|_I \approx 1$, $F|_I \equiv 1$, $\forall X \exists Y\,[F|_I] = 1$. In all examples, I entails F, and "✓" denotes the fact that I passes the test in the column, if applied to the formula in the row.

Consider the first example, $F = (x_1 \vee y \vee x_2)$ and $I = x_1$. We have $F|_I = 1$, and I entails F, which is detected by the syntactic check. For the second example, $F = x_1 y \vee \overline{y} x_2$, we have $F|_I = y \vee \overline{y}$, which is valid, but it syntactically differs from 1. The SAT solver therefore calls Entails on $\neg F \wedge I$. For $\neg F = (\overline{x_1} \vee \overline{y})(y \vee \overline{x_2})$, we find $\neg F|_I = (\overline{y})(y)$. After propagating $\overline{y}$, a conflict at decision level zero occurs, hence Entails returns 1, and an incomplete test is sufficient. In this example, $\neg F$ is already in CNF. The key idea conveyed by it can easily be lifted to the case where additional variables are introduced by the CNF transformation of $\neg F$. For the third example, $F = x_1(\overline{x_2}\,\overline{y} \vee \overline{x_2} y \vee x_2 \overline{y} \vee x_2 y)$, both $P|_I$ and $N|_I$ are undefined and contain no units. However, $N|_I$ is unsatisfiable, the SAT oracle call on $N \wedge I$ terminates with "unsat", and Entails returns 1. Hence, this example requires at least a SAT oracle. For the last example, $F = x_1(x_2 \leftrightarrow y)$, we define

$$P = (x_1)(s_1 \vee s_2)(\overline{s_1} \vee x_2)(\overline{s_1} \vee y)(\overline{s_2} \vee \overline{x_2})(\overline{s_2} \vee \overline{y}) \quad \text{with } S = \{s_1, s_2\} \text{ and}$$
$$N = (\overline{x_1} \vee t_1 \vee t_2)(\overline{t_1} \vee x_2)(\overline{t_1} \vee \overline{y})(\overline{t_2} \vee \overline{x_2})(\overline{t_2} \vee y) \quad \text{with } T = \{t_1, t_2\}$$

We have $P|_I \neq 1$. Neither $P|_I$ nor $N|_I$ contains a unit literal, hence the incomplete test is too weak. Assume a SAT solver is called to check unsatisfiability of $N \wedge I$, and x_2 is decided first. After propagating $\overline{t_2}$, t_1 and $\overline{y}$, a total model of N is found. The SAT solver answers "sat", and Entails returns 0. A QBF solver checking $\varphi = \forall X \exists Y\,[\,x_2 y \vee \overline{x_2}\,\overline{y}\,]$ returns 1. In fact, φ is true for $I = x_2 y$ and $I = \overline{x_2}\,\overline{y}$, and Entails answers 1. Thus, at least a QBF oracle is needed.

EndTrue: $(F,\, I,\, M,\, \delta) \rightsquigarrow_{\mathsf{EndTrue}} M \vee m$ if $V(\mathsf{decs}(I)) \cap X = \emptyset$ and $m \stackrel{\text{def}}{=} \pi(I, X)$ and $\forall X \exists Y\, [\, F|_I \,] = 1$

EndFalse: $(F,\, I,\, M,\, \delta) \rightsquigarrow_{\mathsf{EndFalse}} M$ if exists $C \in F$ and $C|_I = 0$ and $\delta(C) = 0$

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

BackTrue: $(F,\, I,\, M,\, \delta) \rightsquigarrow_{\mathsf{BackTrue}} (F,\, UK\ell,\, M \vee m,\, \delta[L \mapsto \infty][\ell \mapsto b])$ if $UV \stackrel{\text{def}}{=} I$ and $D \stackrel{\text{def}}{=} \overline{\pi(\mathsf{decs}(I), X)}$ and $b + 1 \stackrel{\text{def}}{=} \delta(D) \leqslant \delta(I)$ and $\ell \in D$ and $b = \delta(D \setminus \{\ell\}) = \delta(U)$ and $m \stackrel{\text{def}}{=} \pi(I, X)$ and $K \stackrel{\text{def}}{=} V_{\leqslant b}$ and $L \stackrel{\text{def}}{=} V_{>b}$ and $\forall X \exists Y\, [\, F|_I \,] = 1$

BackFalse: $(F,\, I,\, M,\, \delta) \rightsquigarrow_{\mathsf{BackFalse}} (F,\, UK\ell,\, M,\, \delta[L \mapsto \infty][\ell \mapsto j])$ if exists $C \in F$ and exists D with $UV \stackrel{\text{def}}{=} I$ and $C|_I = 0$ and $c \stackrel{\text{def}}{=} \delta(C) = \delta(D) > 0$ such that $\ell \in D$ and $\bar{\ell} \in \mathsf{decs}(I)$ and $\bar{\ell}|_V = 0$ and $F \wedge \overline{M} \models D$ and $j \stackrel{\text{def}}{=} \delta(D \setminus \{\ell\})$ and $b \stackrel{\text{def}}{=} \delta(U) = c - 1$ and $K \stackrel{\text{def}}{=} V_{\leqslant b}$ and $L \stackrel{\text{def}}{=} V_{>b}$

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

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

Fig. 2. Rules for Enumerate.

5 Formalization

The algorithm listed in Fig. 1 can be expressed by means of a formal calculus. It extends our previous calculus [21] by projection and by a generalized precondition modeling an incremental call to an oracle for checking entailment (lines 17–22 in function Enumerate). Notably, in our work [21], only total models are found, while entailment in our actual work enables the detection of partial models. The variables in Y and S (from the CNF encoding) are treated equally with respect to unit propagation and decision. We therefore merge those two variable sets into Y to simplify the formalization. This does not affect the outcome of the entailment test. In favor of a concise description of the rules, we emphasize the differences to our previous framework [21] and refer to this work for more details.

The procedure terminates as soon as either a conflict at decision level zero occurs (rule EndFalse) or a possibly partial model is found and I contains no

relevant decision literal (rule EndTrue). Requiring that no relevant decision is left on the trail prevents the recording of redundant models. The projection of I onto X is recorded. Rule Unit remains unchanged except for the missing precondition $F|_I \neq 1$. If I entails F and contains relevant decision literals, the one at the highest decision level is flipped, and the projection of I onto X is recorded (rule BackTrue). Requiring that the last relevant decision literal is flipped prevents the recording of redundant models. Rule BackFalse remains unchanged. A decision is taken whenever $F|_I \neq 0$ and $F|_I$ contains no unit. Relevant variables are prioritized (rule DecideX) over irrelevant ones (rule DecideY).

Although not mandatory for correctness, the applicability of rule Unit might be restricted to the case where $F|_I \neq 1$. Similarly, a decision might be taken only if I does not entail F. Notice that in rules Unit, DecideX, and DecideY, the precondition $F|_I \neq 0$ can also be omitted.

6 Conclusion

In many applications (projected) partial models play a central role. For this purpose, we have presented an algorithm and its formalization inspired by how theory solvers and the SAT solver interact in SMT. The basic idea was to detect partial assignments entailing the formula on-the-fly. We presented entailment tests of different strength and computational cost and discussed examples.

The syntactic check "$F|_I = 1$" is cheapest, using clause watches or counters for keeping track of the number of satisfied clauses or alternatively the number of assigned variables (line 11 in Fig. 1). It is also weakest, since $F|_I$ must syntactically coincide with 1. The incomplete check, denoted by "$F|_I \approx 1$", is slightly more involved. It calls a SAT solver on the negation of the formula, restricted, e.g., to unit propagation or a limited number of decision levels, and also might return "unknown". The SAT oracle executes an unsatisfiability check of $\neg F \wedge I$, given a (partial) assignment I, which might be too restrictive. The QBF oracle is the most powerful test, but also the most expensive one. It captures entailment under projection in a precise manner expressed by $\forall X \exists Y\, [\, F|_I \,] = 1$. Combining dual reasoning with oracle calls allows to avoid shrinking of total models. Finally, chronological CDCL renders the use of blocking clauses superfluous.

We claim that this is the first method combining dual reasoning and chronological CDCL for partial model detection. It is anticipated that applications with short partial models benefit most, since oracle calls might be expensive. We plan to implement our method and validate its competitiveness on applications from weighted model integration and model counting with or without projection. We also plan to investigate methods concerning the implementation of QBF oracles required by flavour 4), e.g., dependency schemes introduced by Samer and Szeider [27] or incremental QBF solving proposed by Lonsing and Egly [15].

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 Assocation for Artificial Intelligence (AI*IA). We thank the anonymous reviewers and Mathias Fleury for suggesting many textual improvements.

References

1. Bernasconi, A., Ciriani, V., Luccio, F., Pagli, L.: Compact DSOP and partial DSOP forms. Theory Comput. Syst. **53**(4), 583–608 (2013)
2. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: TACAS. LNCS, vol. 1579, pp. 193–207. Springer (1999)
3. Biere, A., Hölldobler, S., Möhle, S.: An abstract dual propositional model counter. In: YSIP. CEUR Workshop Proceedings, vol. 1837, pp. 17–26. CEUR-WS.org (2017)
4. Brauer, J., King, A., Kriener, J.: Existential quantification as incremental SAT. In: CAV. LNCS, vol. 6806, pp. 191–207. Springer (2011)
5. Davis, M., Logemann, G., Loveland, D.W.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (1962)
6. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**(3), 201–215 (1960)
7. Galindo, J.A., Acher, M., Tirado, J.M., Vidal, C., Baudry, B., Benavides, D.: Exploiting the enumeration of all feature model configurations: a new perspective with distributed computing. In: SPLC. pp. 74–78. ACM (2016)
8. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: *clasp* : A conflict-driven answer set solver. In: LPNMR. LNCS, vol. 4483, pp. 260–265. Springer (2007)
9. Grumberg, O., Schuster, A., Yadgar, A.: Memory efficient all-solutions SAT solver and its application for reachability analysis. In: FMCAD. LNCS, vol. 3312, pp. 275–289. Springer (2004)
10. Gupta, A., Yang, Z., Ashar, P., Gupta, A.: SAT-based image computation with application in reachability analysis. In: FMCAD. LNCS, vol. 1954, pp. 354–371. Springer (2000)
11. Klebanov, V., Manthey, N., Muise, C.J.: SAT-based analysis and quantification of information flow in programs. In: QEST. LNCS, vol. 8054, pp. 177–192. Springer (2013)
12. Lagniez, J., Marquis, P.: A recursive algorithm for projected model counting. In: AAAI. pp. 1536–1543. AAAI Press (2019)
13. Lahiri, S.K., Nieuwenhuis, R., Oliveras, A.: SMT techniques for fast predicate abstraction. In: CAV. LNCS, vol. 4144, pp. 424–437. Springer (2006)
14. Li, B., Hsiao, M.S., Sheng, S.: A novel SAT all-solutions solver for efficient preimage computation. In: DATE. pp. 272–279. IEEE Computer Society (2004)
15. Lonsing, F., Egly, U.: Incremental QBF solving. In: CP. LNCS, vol. 8656, pp. 514–530. Springer (2014)
16. Marques-Silva, J.: Computing with SAT oracles: Past, present and future. In: CiE. LNCS, vol. 10936, pp. 264–276. Springer (2018)
17. Marques-Silva, J.P., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Trans. Computers **48**(5), 506–521 (1999)
18. McMillan, K.L.: Applying SAT methods in unbounded symbolic model checking. In: CAV. LNCS, vol. 2404, pp. 250–264. Springer (2002)
19. Möhle, S., Biere, A.: Dualizing projected model counting. In: Tsoukalas, L.H., Grégoire, É., Alamaniotis, M. (eds.) ICTAI'18. pp. 702–709. IEEE (2018)
20. Möhle, S., Biere, A.: Backing backtracking. In: Janota, M., Lynce, I. (eds.) SAT'19. LNCS, vol. 11628, pp. 250–266. Springer (2019)
21. Möhle, S., Biere, A.: Combining conflict-driven clause learning and chronological backtracking for propositional model counting. In: GCAI. EPiC Series in Computing, vol. 65, pp. 113–126. EasyChair (2019)

22. Morettin, P., Passerini, A., Sebastiani, R.: Efficient weighted model integration via SMT-based predicate abstraction. In: IJCAI. pp. 720–728. ijcai.org (2017)
23. Morettin, P., Passerini, A., Sebastiani, R.: Advanced SMT techniques for weighted model integration. Artif. Intell. **275**, 1–27 (2019)
24. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: DAC. pp. 530–535. ACM (2001)
25. Nadel, A., Ryvchin, V.: Chronological backtracking. In: SAT. LNCS, vol. 10929, pp. 111–121. Springer (2018)
26. Oztok, U., Darwiche, A.: An exhaustive DPLL algorithm for model counting. J. Artif. Intell. Res. **62**, 1–32 (2018)
27. Samer, M., Szeider, S.: Backdoor sets of quantified boolean formulas. J. Autom. Reasoning **42**(1), 77–97 (2009)
28. Sebastiani, R.: Are you satisfied by this partial assignment?, https://arxiv.org/abs/2003.04225.
29. Sebastiani, R.: Lazy Satisfiability Modulo Theories. JSAT **3**(3-4), 141–224 (2007)
30. Sheng, S., Hsiao, M.S.: Efficient preimage computation using a novel success-driven ATPG. In: DATE. pp. 10822–10827. IEEE Computer Society (2003)
31. Strichman, O.: Tuning SAT checkers for bounded model checking. In: CAV. LNCS, vol. 1855, pp. 480–494. Springer (2000)
32. Tibebu, A.T., Fey, G.: Augmenting all solution SAT solving for circuits with structural information. In: DDECS. pp. 117–122. IEEE (2018)
33. Toda, T., Soh, T.: Implementing efficient all solutions SAT solvers. ACM Journal of Experimental Algorithmics **21**(1), 1.12:1–1.12:44 (2016)
34. Zengler, C., Küchlin, W.: Boolean quantifier elimination for automotive configuration - A case study. In: FMICS. LNCS, vol. 8187, pp. 48–62. Springer (2013)