An Abstract CNF-to-d-DNNF Compiler Based on Chronological CDCL

Sibylle Möhle

An Abstract CNF-to-d-DNNF Compiler Based on Chronological CDCL

Sibylle Möhle(✉)

Max Planck Institute for Informatics, Saarland Informatics Campus E1 4, 66123 Saarbrücken, Germany
smoehle@mpi-inf.mpg.de

Abstract. We present Abstract CNF2dDNNF, a calculus describing an approach for compiling a formula in conjunctive normal form (CNF) into deterministic negation normal form (d-DNNF). It combines component-based reasoning with a model enumeration approach based on conflict-driven clause learning (CDCL) with chronological backtracking. Its properties, such as soundness and termination, carry over to implementations which can be modeled by it. We provide a correctness proof and a detailed example. The main conceptual differences to currently available tools targeting d-DNNF compilation are discussed and future research directions presented. The aim of this work is to lay the theoretical foundation for a novel method for d-DNNF compilation. To the best of our knowledge, our approach is the first knowledge compilation method using CDCL with chronological backtracking.

Keywords: Knowledge compilation · d-DNNF · Chronological CDCL

1 Introduction

In real-world applications, constraints may be modeled in conjunctive normal form (CNF), but many tasks relevant in AI and reasoning, such as checks for consistency, validity, clausal entailment, and implicants, can not be executed efficiently on them [9]. Tackling these and other computationally expensive problems is the aim of the knowledge compilation paradigm [13]. The idea is to translate a formula into a language in which the task of interest can be executed efficiently [22]. The knowledge compilation map [22] contains an in-depth discussion of such languages and their properties, and other (families of) languages have been introduced since its publication [21,25,29]. The focus in this work is on the language deterministic decomposable negation normal form (d-DNNF) [19]. It has been applied in planning [2,39], Bayesian reasoning [15], diagnosis [3,43], and machine learning [28] as well as in functional E-MAJSAT [40], to mention a few, and was also studied from a theoretical perspective [7,8,10]. Several d-DNNF compilers are available [20,30,37,48], as well as a d-DNNF reasoner[1].

[1] http://www.cril.univ-artois.fr/kc/d-DNNF-reasoner.html.

U. Sattler and M. Suda (Eds.): FroCoS 2023, LNAI 14279, pp. 195–213, 2023.
https://doi.org/10.1007/978-3-031-43369-6_11

Translating a formula from CNF to d-DNNF requires to process the search space exhaustively. The number of variable assignments which need to be checked is exponential in the number of variables occurring in the formula and testing them one by one is out of question from a computational complexity point of view. However, if the formula can be partitioned into subformulae defined over pairwise disjoint sets of variables, these subformulae can be processed independently and the results combined [4]. This may reduce the amount of work per computation significantly. Consider $F = (a \vee b) \wedge (c \vee d)$ defined over the set of variables $V = \{a, b, c, d\}$. Its search space consists of $2^4 = 16$ variable assignments. The formula F can be partitioned into $F_1 = (a \vee b)$ and $F_2 = (c \vee d)$ defined over the sets of variables $V_1 = \{a, b\}$ and $V_2 = \{c, d\}$, respectively, and such that $F = F_1 \wedge F_2$. Due to $V_1 \cap V_2 = \emptyset$, d-DNNF representations of F_1 and F_2 can be computed independently and conjoined obtaining a d-DNNF representation of F. Moreover, in each computation we only need to check $2^2 = 4$ assignments. The subformulae F_1 and F_2 are called *components* due to the original motivation originating in graph theory, and the partitioning process is referred to as *decomposition* or *component analysis*. This approach, also called *component-based reasoning*, is realized in various exact #SAT solvers [1,4,11,12,41,42,47], and its success suggests that formulae stemming from real-world applications decompose well enough to generate a substantial amount of work saving.

The formula F in our example satisfies *decomposability* [22], i.e., for each conjunction, the conjuncts are defined over pairwise disjoint sets of variables. We call such a formula *decomposable*. Negations occur only in front of literals, hence it is in decomposable negation normal form (DNNF) [17,18]. A formula in which for each disjunction its disjuncts are pairwise logically contradictory satisfies *determinism* [22], i.e., for each disjunction $C_1 \vee \ldots \vee C_n$ it holds that $C_i \wedge C_j \equiv \bot$ for $i, j \in \{1, \ldots, n\}$ and $i \neq j$. A deterministic DNNF formula is said to be in d-DNNF. Determinism is also met by the language disjoint sum of products (DSOP), which is a disjunction of pairwise contradictory conjunctions of literals, and which is relevant in circuit design [5]. In a previous work [34], we introduced an approach for translating a CNF formula into DSOP based on CDCL with chronological backtracking. The motivation for using chronological backtracking is twofold. First, it has shown not to significantly harm solver performance [33,38]. Second, pairwise disjoint models are detected without the need for blocking clauses commonly used in model enumeration based on CDCL with non-chronological backtracking. Blocking clauses rule out already found models, but they also slow down the solver, and avoiding their usage in model enumeration by means of CDCL with chronological backtracking has empirically shown to be effective [46]. Enhancing our former approach [34] by component-based reasoning enables us to compute a d-DNNF representation of a CNF formula. Reconsider our previous example, and suppose we obtained $\mathsf{dsop}(F_1) = a \vee (\neg a \wedge b)$ and $\mathsf{dsop}(F_2) = c \vee (\neg c \wedge d)$. Now $F \equiv F_1 \wedge F_2$, hence $F \equiv \mathsf{dsop}(F_1) \wedge \mathsf{dsop}(F_2) = (a \vee (\neg a \wedge b)) \wedge (c \vee (\neg c \wedge d))$, which is in d-DNNF.

Our Contributions. We present Abstract CNF2dDNNF, ACD for short, a declarative formal framework describing the compilation of CNF into d-DNNF

and a proof of its correctness. This abstract presentation allows for a thorough understanding of our method at a conceptual level and of its correctness. If our framework is sound, every implementation which can be modeled by it is sound as well. This comprises optimizations and implementation details, such as caches. ACD combines component-based reasoning and CNF-to-DSOP compilation based on conflict-driven clause learning (CDCL) with chronological backtracking. Disjunctions with pairwise contradictory disjuncts are introduced by decisions and subsequently flipping their value upon backtracking, while conjunctions whose conjuncts share no variable are introduced by unit propagation and decomposition. For the sake of simplicity, in our calculus formulae are partitioned into two subformulae. However, lifting it to an arbitrary number of subcomponents is straightforward, and a corresponding generalization is presented.

2 Preliminaries

Let V be a set of propositional variables defined over the set of Boolean constants $\bot$ (false) and $\top$ (true) denoted by $\mathbb{B} = \{\bot, \top\}$. A *literal* is either a variable $v \in V$ or its negation $\neg v$. We refer to the variable of a literal ℓ by $\mathsf{var}(\ell)$ and extend this notation to sets and sequences of literals and formulae. We consider formulae in *conjunctive normal form (CNF)* which are conjunctions of *clauses* which are disjunctions of literals. A formula in *disjoint sum of products (DSOP)* is a disjunction of pairwise contradictory *cubes*, which are conjunctions of literals. Our target language is *deterministic decomposable negation normal form (d-DNNF)*, whose formulae are built of literals, conjunctions sharing no variables, and disjunctions whose disjuncts are pairwise contradictory. We might interpret formulae as sets of clauses and cubes and clauses and cubes as sets of literals by writing $C \in F$ and $\ell \in C$ to refer to a clause C in a formula F and a literal ℓ contained in a clause or cube C, respectively. The empty CNF formula and the empty cube are denoted by $\top$ and the empty DSOP formula and the empty clause by $\bot$.

A *total variable assignment* is a mapping $\sigma \colon V \mapsto \mathbb{B}$, and 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, such that $I(\ell) = \top$ iff $\ell \in I$. Similarly, $I(C)$ and $I(F)$ are defined. We might interpret a trail I as a set of literals and write $\ell \in I$ to refer to the literal ℓ on I. The empty trail is denoted by ε and the set of variables of the literals on I by $\mathsf{var}(I)$. Trails and literals can be concatenated, written $I\,J$ and $I\,\ell$, given $\mathsf{var}(I) \cap \mathsf{var}(J) = \emptyset$ and $\mathsf{var}(I) \cap \mathsf{var}(\ell) = \emptyset$. The position of ℓ on the trail I is denoted by $\tau(I, \ell)$. The decision literals on I are annotated by a superscript, e.g., ℓ^d, denoting open "left" branches in the sense of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [23,24]. *Flipping the value of a decision literal* can be seen as closing the corresponding left branch and starting a "right" branch, where the decision literal ℓ^d becomes a *flipped literal* $\neg\ell$.

The *residual* of F under I, written $F|_I$, is obtained by assigning the variables in F their truth value and by propagating truth values through Boolean

connectives. The notion of residual is extended to clauses and literals. A *unit clause* is a clause $\{\ell\}$ containing one single literal ℓ. By $\mathsf{units}(F)$ ($\mathsf{units}(F|_I)$) we denote the set of unit literals in F ($F|_I$). Similarly, $\mathsf{decs}(I)$ denotes the set of decision literals on I. By writing $\ell \in \mathsf{decs}(I)$ ($\ell \in \mathsf{units}(F)$, $\ell \in \mathsf{units}(F|_I)$), we refer to a decision literal ℓ on I (unit literal in F, $F|_I$). A trail I *falsifies* F, if $I(F) \equiv \bot$, i.e., $F|_I = \bot$. It *satisfies* F, $I \models F$, if $I(F) \equiv \top$, i.e., $F|_I = \top$, and is then called a *model* of F. If $\mathsf{var}(I) = V$, I is a *total model*, otherwise it is a *partial model*.

The trail is partitioned into *decision levels*, starting with a decision literal and extending until the literal preceding the next decision. The *decision level function* $\delta\colon V \mapsto \mathbb{N} \cup \{\infty\}$ returns the decision level of a variable $v \in V$. If v is unassigned, $\delta(v) = \infty$, and δ is updated whenever a variable is assigned or unassigned, e.g., $\delta[v \mapsto d]$ if v is assigned to decision level d. We define $\delta(\ell) = \delta(\mathsf{var}(\ell))$, $\delta(C) = \mathsf{max}\{\delta(\ell) \mid \ell \in C\}$ for $C \neq \bot$ and $\delta(I) = \mathsf{max}\{\delta(\ell) \mid \ell \in I\}$ for $I \neq \varepsilon$ extending this notation to sets of literals. Finally, we define $\delta(\bot) = \delta(\varepsilon) = \infty$. By writing $\delta[I \mapsto \infty]$, all literals on the trail I are unassigned. The decision level function is left-associative, i.e., $\delta[I \mapsto \infty][\ell \mapsto d]$ expresses that first all literals on I are unassigned and then literal ℓ is assigned to decision level d.

Unlike in CDCL with non-chronological backtracking [36,44,45], in *chronological CDCL* [33,38] literals may not be ordered on the trail in ascending order with respect to their decision level. We write $I_{\leqslant n}$ ($I_{<n}$, $I_{=n}$) for the subsequence of I containing all literals ℓ with $\delta(\ell) \leq n$ ($\delta(\ell) < n$, $\delta(\ell) = n$). The *pending search space* of I is given by the assignments not yet tested [34], i.e., I and its open right branches $R(I)$, and is defined as $O(I) = I \vee R(I)$, where $R(I) = \bigvee_{\ell \in \mathsf{decs}(I)} R_{=\delta(\ell)}(I)$ and $R_{=\delta(\ell)}(I) = \neg\ell \wedge I_{<\delta(\ell)}$ for $\ell \in \mathsf{decs}(I)$. As an example, for $I = a\,b^d\,c\,d\,e^d\,f$, $O(I) = (a \wedge b \wedge c \wedge d \wedge e \wedge f) \vee (\neg b \wedge a) \vee (\neg e \wedge a \wedge b \wedge c \wedge d)$. Similarly, the *pending models* of F are the satisfying assignments of F not yet detected and which are given by $F \wedge O(I)$.

3 Chronological CDCL for CNF-to-d-DNNF Compilation

In *static component analysis* the component structure is computed once, typically as a preprocessing step, and not altered during the further execution. In contrast, in our approach the component structure is computed iteratively adopting *dynamic component analysis*. Algorithm 1 provides a general schema in pseudo-code. It is formulated recursively, capturing the recursive nature of dynamic component analysis. Lines 1–7 and 11 describe model enumeration based on chronological CDCL [34], while lines 8–10 capture component analysis.

Now assume unit propagation has been carried out until completion, no conflict has occurred and there are still unassigned variables (line 8). If $F|_I$ can be decomposed into two formulae G and H, we call $\mathsf{CNF2dDNNF}$ recursively on G and H, conjoin the outcomes of these computations with I and add the result to M (line 9). If I contains no decisions, the search space has been explored exhaustively, otherwise chronological backtracking occurs (lines 10). The working of our approach is shown by an example.

Algorithm 1: CNF2dDNNF(F, V, I, M)

input : CNF $F, V = \mathsf{var}(F), I = \varepsilon, M = \bot$
output: d-DNNF $M \equiv F$

Loop
  $I \leftarrow$ PropagateUnits()
  if *conflict occurs* **then**
    if *conflict level* $= 0$ **then return** M **else** AnalyzeConflict()
  else if $I(F) = \top$ **then**
    $M \leftarrow M \vee I$
    if *there are no decisions on* I **then return** M **else** BacktrackChrono()
  else if $F|_I$ *can be decomposed into* G *and* H **then**
    $M \leftarrow M \vee I \wedge$ CNF2dDNNF$(G, \mathsf{var}(G), \varepsilon, \bot) \wedge$ CNF2dDNNF$(H, \mathsf{var}(H), \varepsilon, \bot)$
    if *there are no decisions on* I **then return** M **else** BacktrackChrono()
  else Decide

Example 1. Let $V = \{a, b, c, d, e, f, g, h\}$ be a set of propositional variables and $F = (a) \wedge (\neg a \vee \neg b \vee c \vee d) \wedge (\neg a \vee \neg b \vee e \vee f) \wedge (b \vee \neg c \vee e) \wedge (b \vee d \vee f) \wedge (g \vee h)$ be a formula defined over V. The execution is depicted as a tree in Fig. 1. For the sake of readability, we show only the formula on which a rule is executed, represented by a box annotated with its component level. Black arrows correspond to "downward" rule applications, while violet (gray) arrows represent "upwards" rule applications and are annotated with the formula returned by the computation of a component. Ignore the rule names for now, they are intended to clarify the working of our calculus which is presented in Sect. 4. We see that, first, a is propagated, denoted by the black vertical arrow annotated with a and the name of the applied rule (Unit). The residual of F under a is $F|_a = (\neg b \vee c \vee d) \wedge (\neg b \vee e \vee f) \wedge (b \vee \neg c \vee e) \wedge (b \vee d \vee f) \wedge (g \vee h)$ (not shown). It contains no unit clause but can be decomposed into $(\neg b \vee c \vee d) \wedge (\neg b \vee e \vee f) \wedge (b \vee \neg c \vee e) \wedge (b \vee d \vee f)$ and $(g \vee h)$. Two new (sub)components are created (by applying rule Decompose) with component level 01 and 02, respectively, represented by the shadowed boxes.

Since $(g \vee h)$ can not be decomposed further, model enumeration with chronological CDCL is executed on it (not shown) by deciding g (rule Decide) satisfying $(g \vee h)$, followed by backtracking chronologically (BackTrue), which amounts to negating the value of the most recent decision g, and propagating h (Unit). The processing of $(g \vee h)$ terminates with $g \vee \neg g \wedge h$ (CompTrue, not shown). But before this result can be used further, the subcomponent at component level 01 needs to be processed. Its formula is $G = (\neg b \vee c \vee d) \wedge (\neg b \vee e \vee f) \wedge (b \vee \neg c \vee e) \wedge (b \vee d \vee f)$. It neither contains a unit nor can it be decomposed, hence we take a decision, let's say, b^d. Now $G|_b = (c \vee d) \wedge (e \vee f)$, which is decomposed into two components with one clause each and component level 011 and 012, respectively (Decompose). These formulae can not be decomposed further, and they are processed independently, similarly to $(g \vee h)$. Before G was decomposed, a decision was taken, and we backtrack combining the results of its subcomponents (ComposeBack). We have $G|_{\neg b} = (\neg c \vee e) \wedge (d \vee f)$ resulting in two components with component

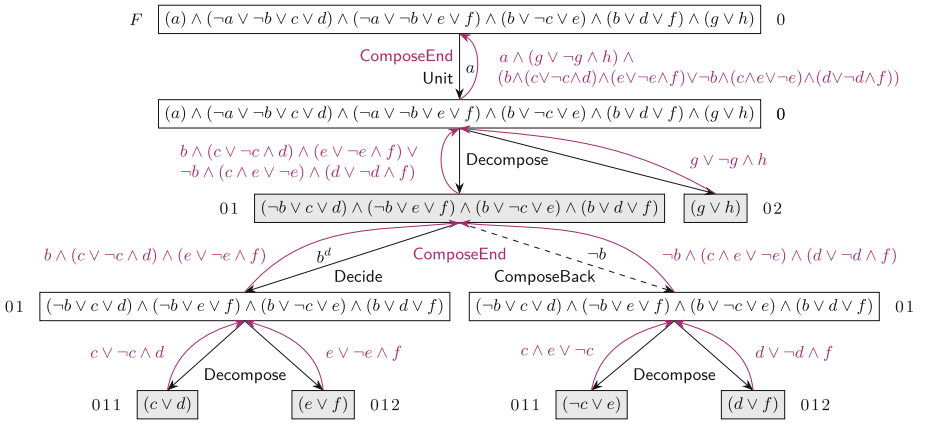

Fig. 1. Component structure of F created by ACD.

levels 011 and 012, respectively. They are processed and their results combined, after which the results of the subcomponents of the root component are conjoined with a. There is no decision on the trail, and the process terminates with $M = (a) \wedge (\neg a \vee \neg b \vee c \vee d) \wedge (\neg a \vee \neg b \vee e \vee f) \wedge (b \vee \neg c \vee e) \wedge (b \vee d \vee f) \wedge (g \vee h)$ (ComposeEnd). Notice that although component levels can occur multiple times throughout the computation, they are unique at any point in time.

4 Calculus

Due to its recursive nature, combining the results computed for subcomponents in CNF2dDNNF is straightforward. For its formalization, however, a non-recursive approach turned out to be better suited. Consequently, a method is needed for matching subcomponents and their parent. For this purpose, a *component level* is associated with each component. It is defined as a string of numbers in $\mathbb{N}$ as follows. Suppose a component $\mathcal{C}$ is assigned level "d" and assume its formula is decomposed into two subformulae. The corresponding subcomponents $\mathcal{C}_G$ and $\mathcal{C}_H$ are assigned component levels "$d \cdot 1$" and "$d \cdot 2$", respectively, with "$\cdot$" denoting string composition. Accordingly, the component level of their parent $\mathcal{C}$ is given by the substring consisting of all but the last element of their level, i.e., "d".[2] The *root component* holds the input formula, it has no parent and its component level is zero. A component is *closed* if no rule can be applied to it, and *decomposed* if either at least one of its subcomponents is not closed or both its subcomponents are closed, but their results are not yet combined. Components which are neither closed nor decomposed are *open*.[3] Closed components may be discarded as soon

[2] From now on, we omit the quotes for the sake of readability.

[3] The differentiation between open and decomposed components is purely technical and needed for the termination proof in Sect. 5.

as their results are combined, and the computation stops as soon as the root component is closed. With these remarks, we are ready to present our calculus.

We describe our algorithm in terms of a state transition system ABSTRACT CNF2DDNNF, ACD for short, over a set of global states S, a transition relation $\rightsquigarrow \subseteq S \times S$ and an initial global state $\mathcal{S}_0$. A *global state* is a set of components. A *component* $\mathcal{C}$ is described as a seven-tuple $(F, V, d, e, I, M, \delta)^s$, where s denotes its *component state*. It is c if $\mathcal{C}$ is closed, f if F is decomposed, and o if $\mathcal{C}$ is open. The first two elements F and V refer to a formula and its set of variables, respectively. The third element d denotes the component level of $\mathcal{C}$. If $d \neq 0$, then $d \in \{d' \cdot 1, d' \cdot 2\}$, where d' is the component level of the parent component of $\mathcal{C}$, as explained above. In this manner, the component level keeps track of the decomposition structure of F and is used to match parent components and their subcomponents. The number of subcomponents of $\mathcal{C}$ is given by e, while I and δ refer to a trail ranging over variables in V and a decision level function with domain V, respectively. Finally, M is a formula in d-DNNF representing the models of F found so far. A component is initialized by $(F, V, d, 0, \varepsilon, \bot, \infty)^o$ and closed after its computation has terminated, i.e., $(F, V, d, 0, I, M, \delta)^c$. Notice that in these cases $e = 0$. The *initial global state* $\mathcal{S}_0 = \{\mathcal{C}_0\}$ consists of the *root component* $\mathcal{C}_0 = (F, V, 0, 0, \varepsilon, \bot, \infty)^o$ with F and V denoting the input formula and $V = \mathsf{var}(F)$, while the *final global state* is given by $\mathcal{S}_n = \{(F, V, 0, 0, I, M, \delta)^c\}$ where $M \equiv F$ is in d-DNNF. The transition relation $\rightsquigarrow$ is defined as the union of transition relations $\rightsquigarrow_{\mathsf{R}}$, where R is either Unit, Decide, BackTrue, BackFalse, CompTrue, CompFalse, Decompose, ComposeBack or ComposeEnd. Our calculus contains three types of rules, which can abstractly be described as follows:

$$\alpha\colon \mathcal{S} \uplus \{\mathcal{C}\} \rightsquigarrow_{\mathsf{R}} \mathcal{S} \uplus \mathcal{C}'; \; \beta\colon \mathcal{S} \uplus \{\mathcal{C}\} \rightsquigarrow_{\mathsf{R}} \mathcal{S} \uplus \{\mathcal{C}', \mathcal{C}_1, \mathcal{C}_2\}; \; \gamma\colon \mathcal{S} \uplus \{\mathcal{C}, \mathcal{C}_1, \mathcal{C}_2\} \rightsquigarrow_{\mathsf{R}} \mathcal{S} \uplus \{\mathcal{C}'\}.$$

In this description, $\mathcal{S}$ refers to the subset of the current global state consisting of all components which are not touched by rule R, with $\uplus$ denoting the disjoint set union, e.g., in α, $\mathcal{C}, \mathcal{C}' \notin \mathcal{S}$. An α rule affects a component $\mathcal{C}$ turning it into $\mathcal{C}'$. The rules Unit, Decide, BackTrue, BackFalse, CompTrue, and CompFalse are α rules. A β rule modifies $\mathcal{C}$ obtaining $\mathcal{C}'$ and creates two new components $\mathcal{C}_1$ and $\mathcal{C}_2$. Rule Decompose is the only β rule. Finally, a γ rule removes the two components $\mathcal{C}_1$ and $\mathcal{C}_2$ from the global state and modifies their parent $\mathcal{C}$. Rules ComposeBack and ComposeEnd are γ rules. The rules are listed in Fig. 2.

Model Computation. Rules Unit, Decide, BackTrue, BackFalse, CompTrue, and CompFalse execute model enumeration with chronological CDCL [34] and are applicable exclusively to open components. Unit literals are assigned the decision level of their reason, which might be lower than the current decision level (rule Unit). Decisions can be taken only if the processed formula is not decomposable (Decide). Backtracking occurs chronologically, i.e., to the second highest decision level on the trail, after finding a model (BackTrue) and to the decision level preceding the conflict level after conflict analysis (BackFalse), respectively. In the latter case, the propagated literal is assigned the lowest level at which the learned clause becomes unit and to which a SAT solver implementing CDCL with

Unit:	$\mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^o\} \rightsquigarrow_{\mathsf{Unit}} \mathcal{S} \uplus \{(F, V, d, 0, I\,\ell, M, \delta[\ell \mapsto a])^o\}$ if $\bot \notin F|_I$ and exists $C \in F$ with $\{\ell\} = C|_I$ and $a \stackrel{\text{def}}{=} \delta(C \setminus \{\ell\})$ if $C \setminus \{\ell\} \neq \bot$ and $a \stackrel{\text{def}}{=} 0$ otherwise
Decide:	$\mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^o\} \rightsquigarrow_{\mathsf{Decide}} \mathcal{S} \uplus \{(F, V, d, 0, I\,\ell^d, M, \delta[\ell \mapsto a])^o\}$ if $F|_I \neq \top$ and $\bot \notin F|_I$ and $\mathsf{units}(F|_I) = \emptyset$ and $\mathsf{var}(\ell) \in V$ and $\delta(\ell) = \infty$ and $a \stackrel{\text{def}}{=} \delta(I) + 1$ and there exist no G and H such that $G \wedge H = F|_I$ and $\mathsf{var}(G) \cap \mathsf{var}(H) = \emptyset$
BackTrue:	$\mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^o\} \rightsquigarrow_{\mathsf{BackTrue}}$ $\mathcal{S} \uplus \{(F, V, d, 0, PK\ell, M \vee I, \delta[L \mapsto \infty][\ell \mapsto e])^o\}$ if $F|_I = \top$ and $PQ \stackrel{\text{def}}{=} I$ and $D \stackrel{\text{def}}{=} \neg\mathsf{decs}(I)$ and $e + 1 \stackrel{\text{def}}{=} \delta(D) = \delta(I)$ and $\ell \in D$ and $e = \delta(D \setminus \{\ell\}) = \delta(P)$ and $K \stackrel{\text{def}}{=} Q_{\leqslant e}$ and $L \stackrel{\text{def}}{=} Q_{>e}$
BackFalse:	$\mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^o\} \rightsquigarrow_{\mathsf{BackFalse}}$ $\mathcal{S} \uplus \{(F, V, d, 0, PK\ell, M, \delta[L \mapsto \infty][\ell \mapsto j])^o\}$ if exists $C \in F$ and exists D with $PQ \stackrel{\text{def}}{=} I$ and $C|_I = \bot$ and $c \stackrel{\text{def}}{=} \delta(C) = \delta(D) > 0$ such that $\ell \in D$ and $\neg\ell \in \mathsf{decs}(I)$ and $\ell|_Q = \bot$ and $F \wedge \neg M \models D$ and $j \stackrel{\text{def}}{=} \delta(D \setminus \{\ell\})$ and $b \stackrel{\text{def}}{=} \delta(P) = c - 1$ and $K \stackrel{\text{def}}{=} Q_{\leqslant b}$ and $L \stackrel{\text{def}}{=} Q_{>b}$
CompTrue:	$\mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^o\} \rightsquigarrow_{\mathsf{CompTrue}} \mathcal{S} \uplus \{(F, V, d, 0, I, M \vee I, \delta)^c\}$ if $F|_I = \top$ and $\mathsf{decs}(I) = \emptyset$
CompFalse:	$\mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^o\} \rightsquigarrow_{\mathsf{CompFalse}} \mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^c\}$ if exists $C \in F$ and $C|_I = \bot$ and $\delta(C) = 0$
Decompose:	$\mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^o\} \rightsquigarrow_{\mathsf{Decompose}}$ $\mathcal{S} \uplus \{(F, V, d, 2, I, M, \delta)^f, (G, U, d{\cdot}1, 0, \varepsilon, \bot, \infty)^o, (H, W, d{\cdot}2, 0, \varepsilon, \bot, \infty)^o\}$ if $F|_I \neq \top$ and $\bot \notin F|_I$ and $\mathsf{units}(F|_I) = \emptyset$ and $G \wedge H \stackrel{\text{def}}{=} F|_I$ and $U \stackrel{\text{def}}{=} \mathsf{var}(G)$ and $W \stackrel{\text{def}}{=} \mathsf{var}(H)$ and $U \cap W = \emptyset$
ComposeBack:	$\mathcal{S} \uplus \{(F, V, d, 2, I, M, \delta_F)^f,$ $(G, U, d \cdot 1, 0, J_G, N, \delta_G)^c, (H, W, d \cdot 2, 0, J_H, O, \delta_H)^c\} \rightsquigarrow_{\mathsf{ComposeBack}}$ $\mathcal{S} \uplus \{(F, V, d, 0, PK\ell, M \vee (I \wedge N \wedge O), \delta[L \mapsto \infty][\ell \mapsto e])^o\}$ if $PQ \stackrel{\text{def}}{=} I$ and $D \stackrel{\text{def}}{=} \neg\mathsf{decs}(I)$ and $e + 1 \stackrel{\text{def}}{=} \delta(D) = \delta(I)$ and $\ell \in D$ and $e = \delta(D \setminus \{\ell\}) = \delta(P)$ and $K \stackrel{\text{def}}{=} Q_{\leqslant e}$ and $L \stackrel{\text{def}}{=} Q_{>e}$
ComposeEnd:	$\mathcal{S} \uplus \{(F, V, d, 2, I, M, \delta)^f,$ $(G, U, d \cdot 1, 0, J_G, N, \delta_G)^c, (H, W, d \cdot 2, 0, J_H, O, \delta_H)^c\} \rightsquigarrow_{\mathsf{ComposeEnd}}$ $\mathcal{S} \uplus \{(F, V, d, 0, I, M \vee (I \wedge N \wedge O), \delta)^c\}$ if $\mathsf{decs}(I) = \emptyset$

Fig. 2. ACD transition rules.

non-chronological backtracking would backtrack to. Since the literals might not be ordered on the trail in ascending order with respect to their decision level, a non-contiguous part of it is discarded. Finally, a component is closed if its trail contains no decisions and either satisfies its formula (CompTrue) or a conflict occurs at decision level zero, i.e., the conflicting clause has decision level zero (CompFalse). In the former case, the newly found model is recorded.

Component Analysis. Rules Decompose, ComposeBack, and ComposeEnd capture the decomposition of a formula and the combination of the models of its subformulae and thus affect multiple components.

Decompose. The state of the parent component $\mathcal{C}$ with formula F is o (open). The trail I neither satisfies nor falsifies F, and $F|_I$ contains no unit clause but can

be partitioned into two formulae G and H defined over disjoint sets of variables. Subcomponents for G and H are created, the number of subcomponents of $\mathcal{C}$ is set to two and its state is changed to f (decomposed). Notice that $\mathcal{C}$ can only be processed further after its subcomponents are closed.

ComposeBack. The state of the component $\mathcal{C}$ with formula F is f (decomposed). Its subcomponents $\mathcal{C}_G$ and $\mathcal{C}_H$ with formulae G and H, respectively, have state c (closed). Furthermore, $N \equiv G$ and $O \equiv H$, hence $F|_I \equiv I \wedge N \wedge O$, which is added to M. This corresponds to enumerating multiple models of F in one step. This can easily be seen by applying the distributive laws to $I \wedge N \wedge O$ which gives us a DSOP formula whose disjuncts are satisfying assignments of $F|_I$. The search space has not yet been processed exhaustively ($\delta(I) > 0$), backtracking to the second highest decision level occurs, and the state of $\mathcal{C}$ is changed back to o (open). Finally, $\mathcal{C}_G$ and $\mathcal{C}_H$ are removed from the global state. If I can not be extended to a model of F, we have $N = \bot$ or $O = \bot$, and $I \wedge N \wedge O = \bot$. Otherwise, $I \wedge N \wedge O \neq \bot$. Both cases are captured by rule ComposeBack.

ComposeEnd. The state of the parent component $\mathcal{C}$ with formula F is f (decomposed). Its subcomponents $\mathcal{C}_G$ and $\mathcal{C}_H$ with formulae G and H, respectively, are closed. Furthermore, $N \equiv G$ and $O \equiv H$, hence $F|_I \equiv I \wedge N \wedge O$, which is added to M. The search space has been processed exhaustively ($\mathsf{decs}(I) = \emptyset$), and the state of $\mathcal{C}$ is set to c (closed). Finally, $\mathcal{C}_G$ and $\mathcal{C}_H$ are removed from the global state. As in rule ComposeBack, either $I \wedge N \wedge O = \bot$ or $I \wedge N \wedge O \neq \bot$.

Example 2. Reconsider Example 1 with variables $V = \{a, b, c, d, e, f, g, h\}$ and $F = (a) \wedge (\neg a \vee \neg b \vee c \vee d) \wedge (\neg a \vee \neg b \vee e \vee f) \wedge (b \vee \neg c \vee e) \wedge (b \vee d \vee f) \wedge (g \vee h)$ defined over V. The execution trace of ACD is shown in Fig. 3. Unaffected components are depicted in gray, and model enumeration by means of chronological CDCL is shown only once in full detail. The execution starts with the root component $\mathcal{C}_F$ containing F. In step (1), the unit literal a is propagated, upon which $F|_a$ is decomposed into $(g \vee h)$ and G creating components $\mathcal{C}_{(g \vee h)}$ and $\mathcal{C}_G$ shown in (2). Steps (3) to (6) capture model enumeration by chronological CDCL of $(g \vee h)$, i.e., the computation of a DSOP representation of $(g \vee h)$, after which $\mathcal{C}_{(g \vee h)}$ is closed. Next, the formula G is processed by deciding b in step (7), decomposing $G|_b$ into $(c \vee d)$ and $(e \vee f)$ and creating components $\mathcal{C}_{(c \vee d)}$ and $\mathcal{C}_{(e \vee f)}$, respectively, in step (8). The processing of $\mathcal{C}_{(c \vee d)}$ and $\mathcal{C}_{(e \vee f)}$ occurs analogously to steps (3) to (6) resulting in the state shown in (9). The results are conjoined with b, which is the trail of $\mathcal{C}_G$ and under which $G|_b$ was decomposed. Since b is a decision, it is flipped in (10) to explore its right branch $\neg b$. The formula $G|_{\neg b}$ is decomposed into $(\neg c \vee e)$ and $(d \vee f)$ and components $\mathcal{C}_{(\neg c \vee e)}$ and $\mathcal{C}_{(d \vee f)}$ are created, as in (11). Their processing, which is not shown, results in the state depicted in (12), and the results are conjoined with the trail of $\mathcal{C}_G$. Since its trail contains no decision, $\mathcal{C}_G$ is closed, see (13). The global state now contains the root component and its two subcomponents, which are closed, hence the rule ComposeEnd is executed, and the computation terminates with the closed root component and $M = a \wedge (g \vee \neg g \wedge h) \wedge (b \wedge (c \vee \neg c \wedge d) \wedge (e \vee \neg e \wedge f) \vee \neg b \wedge (c \wedge e \vee \neg c) \wedge (d \vee \neg d \wedge f)$, where $M \equiv F$, and which is shown in (14).

$\{(F,\, V,\, 0,\, 0,\, \varepsilon,\, \bot,\, \delta_{00} = \infty)^o\}$

$\rightsquigarrow_{\mathsf{Unit}}$ $\{(F,\, V,\, 0,\, 0,\, a,\, \bot,\, \delta_{01} = \delta_{00}[a \mapsto 0])^o\}$ (1)

$\rightsquigarrow_{\mathsf{Decompose}}$ $\{(F,\, V,\, 0,\, 2,\, a,\, \bot,\, \delta_{01})^f,\, ((g \vee h),\, \{g, h\},\, 02,\, 0,\, \varepsilon,\, \bot,\, \delta_{020} = \infty)^o,$ (2)
$(G = (\neg b \vee c \vee d) \wedge (\neg b \vee e \vee f) \wedge (b \vee \neg c \vee e) \wedge (b \vee d \vee f),\, \{b, c, d, e, f\},\, 01,\, 0,\, \varepsilon,\, \bot,\, \delta_{010} = \infty)^o\}$

// enumerate models of $(g \vee h)$ with chronological CDCL

$\rightsquigarrow_{\mathsf{Decide}}$ $\{(F,\, V,\, 0,\, 2,\, a,\, \bot,\, \delta_{01})^f,\, ((g \vee h),\, \{g, h\},\, 02,\, 0,\, g^d,\, \bot,\, \delta_{021} = \delta_{020}[g \mapsto 1])^o,$ (3)
$(G,\, \{b, c, d, e, f\},\, 01,\, 0,\, \varepsilon,\, \bot,\, \delta_{010})^o\}$

$\rightsquigarrow_{\mathsf{BackTrue}}$ $\{(F,\, V,\, 0,\, 2,\, a,\, \bot,\, \delta_{01})^f,\, ((g \vee h),\, \{g, h\},\, 02,\, 0,\, \neg g,\, g,\, \delta_{022} = \delta_{021}[g \mapsto \infty][g \mapsto 0])^o,$ (4)
$(G,\, \{b, c, d, e, f\},\, 01,\, 0,\, \varepsilon,\, \bot,\, \delta_{010})^o\}$

$\rightsquigarrow_{\mathsf{Unit}}$ $\{(F,\, V,\, 0,\, 2,\, a,\, \bot,\, \delta_{01})^f,\, ((g \vee h),\, \{g, h\},\, 02,\, 0,\, \neg g h,\, g,\, \delta_{023} = \delta_{022}[h \mapsto 0])^o,$ (5)
$(G,\, \{b, c, d, e, f\},\, 01,\, 0,\, \varepsilon,\, \bot,\, \delta_{010})^o\}$

$\rightsquigarrow_{\mathsf{CompTrue}}$ $\{(F,\, V,\, 0,\, 2,\, a,\, \bot,\, \delta_{01})^f,\, ((g \vee h),\, \{g, h\},\, 02,\, 0,\, \neg g h,\, g \vee \neg g \wedge h,\, \delta_{023})^c,$ (6)
$(G,\, \{b, c, d, e, f\},\, 01,\, 0,\, \varepsilon,\, \bot,\, \delta_{010})^o\}$

$\rightsquigarrow_{\mathsf{Decide}}$ $\{(F,\, V,\, 0,\, 2,\, a,\, \bot,\, \delta_{01})^f,\, ((g \vee h),\, \{g, h\},\, 02,\, 0,\, \neg g h,\, g \vee \neg g \wedge h,\, \delta_{023})^c,$ (7)
$(G,\, \{b, c, d, e, f\},\, 01,\, 0,\, b^d,\, \bot,\, \delta_{011} = \delta_{010}[b \mapsto 1])^o\}$

$\rightsquigarrow_{\mathsf{Decompose}}$ $\{(F,\, V,\, 0,\, 2,\, a,\, \bot,\, \delta_{01})^f,\, ((g \vee h),\, \{g, h\},\, 02,\, 0,\, \neg g h,\, g \vee \neg g \wedge h,\, \delta_{023})^c,$ (8)
$(G,\, \{b, c, d, e, f\},\, 01,\, 2,\, b^d,\, \bot,\, \delta_{011})^f,$
$((c \vee d),\, \{c, d\},\, 011,\, 0,\, \varepsilon,\, \bot,\, \delta_{0110} = \infty)^o,\, ((e \vee f),\, \{e, f\},\, 012,\, 0,\, \varepsilon,\, \bot,\, \delta_{0120} = \infty)^o\}$

// enumerate models of $(c \vee d)$ and $(e \vee f)$ with chronological CDCL

...

$\rightsquigarrow_{\mathsf{CompTrue}}$ $\{(F,\, V,\, 0,\, 2,\, a,\, \bot,\, \delta_{01})^f,\, ((g \vee h),\, \{g, h\},\, 02,\, 0,\, \neg g h,\, g \vee \neg g \wedge h,\, \delta_{023})^c,$ (9)
$(G,\, \{b, c, d, e, f\},\, 01,\, 2,\, b^d,\, \bot,\, \delta_{011})^f,$
$((c \vee d),\, \{c, d\},\, 011,\, 0,\, \neg c d,\, c \vee \neg c \wedge d,\, \delta_{0113})^c,\, ((e \vee f),\, \{e, f\},\, 012,\, 0,\, \neg e f,\, e \vee \neg e \wedge f,\, \delta_{0123})^c\}$

// combine results

$\rightsquigarrow_{\mathsf{ComposeBack}}$ $\{(F,\, V,\, 0,\, 2,\, a,\, \bot,\, \delta_{01})^f,\, ((g \vee h),\, \{g, h\},\, 02,\, 0,\, \neg g h,\, g \vee \neg g \wedge h,\, \delta_{023})^c,$ (10)
$(G,\, \{b, c, d, e, f\},\, 01,\, 0,\, \neg b,\, b \wedge (c \vee \neg c \wedge d) \wedge (e \vee \neg e \wedge f),\, \delta_{012} = \delta_{011}[b \mapsto \infty][b \mapsto 0])^o\}$

$\rightsquigarrow_{\mathsf{Decompose}}$ $\{(F,\, V,\, 0,\, 2,\, a,\, \bot,\, \delta_{01})^f,\, ((g \vee h),\, \{g, h\},\, 02,\, 0,\, \neg g h,\, g \vee \neg g \wedge h,\, \delta_{023})^c,$ (11)
$(G,\, \{b, c, d, e, f\},\, 01,\, 2,\, \neg b,\, b \wedge (c \vee \neg c \wedge d) \wedge (e \vee \neg e \wedge f),\, \delta_{012})^f,$
$((\neg c \vee e),\, \{c, e\},\, 011,\, 0,\, \varepsilon,\, \bot,\, \delta_{0110} = \infty)^o,((d \vee f),\, \{d, f\},\, 012,\, 0,\, \varepsilon,\, \bot,\, \delta_{0120} = \infty)^o$

// enumerate models of $(\neg c \vee e)$ and $(d \vee f)$ with chronological CDCL

...

$\rightsquigarrow_{\mathsf{CompTrue}}$ $\{(F,\, V,\, 0,\, 2,\, a,\, \bot,\, \delta_{01})^f,\, ((g \vee h),\, \{g, h\},\, 02,\, 0,\, \neg g h,\, g \vee \neg g \wedge h,\, \delta_{023})^c,$ (12)
$(G,\, \{b, c, d, e, f\},\, 01,\, 2,\, \neg b,\, b \wedge (c \vee \neg c \wedge d) \wedge (e \vee \neg e \wedge f),\, \delta_{012})^f,$
$((\neg c \vee e),\, \{c, e\},\, 011,\, 0,\, \neg c,\, c \wedge e \vee \neg c,\, \delta_{0113})^c,\, ((d \vee f),\, \{d, f\},\, 012,\, 0,\, \neg d f,\, d \vee \neg d \wedge f,\, \delta_{0123})^c\}$

// combine results

$\rightsquigarrow_{\mathsf{ComposeEnd}}$ $\{(F,\, V,\, 0,\, 2,\, a,\, \bot,\, \delta_{01})^f,\, ((g \vee h),\, \{g, h\},\, 02,\, 0,\, \neg g h,\, g \vee \neg g \wedge h,\, \delta_{023})^c,$ (13)
$(G,\, \{b, c, d, e, f\},\, 01,\, 2,\, \neg b,\, b \wedge (c \vee \neg c \wedge d) \wedge (e \vee \neg e \wedge f) \vee \neg b \wedge (c \wedge e \vee \neg c) \wedge (d \vee \neg d \wedge f),\, \delta_{012})^c\}$

$\rightsquigarrow_{\mathsf{ComposeEnd}}$ $\{(F,\, V,\, 0,\, 2,\, a,\, a \wedge (g \vee \neg g \wedge h) \wedge (b \wedge (c \vee \neg c \wedge d) \wedge (e \vee \neg e \wedge f) \vee \neg b \wedge (c \wedge e \vee \neg c) \wedge (d \vee \neg d \wedge f),\, \delta_{012})^c\}$ (14)

Fig. 3. Execution trace of ACD for Example 1.

5 Proofs

For proving correctness, we first show that our calculus is sound by identifying invariants which need to hold in a sound global state and show that they still hold after the execution of any rule. Then we prove that for any closed component it holds that $M \equiv F$ and that ACD can not get stuck and terminates in a correct state. Showing termination concludes our proof.

Definition 1 (Sound Global State). *A global state $\mathcal{S}$ is* sound *if for all its components $\mathcal{C} = (F,\, V,\, d,\, e,\, I,\, M,\, \delta)^s$ the following invariants hold:*

(1) $\forall k, \ell \in \mathsf{decs}(I)\,.\, \tau(I, k) < \tau(I, \ell) \implies \delta(k) < \delta(\ell)$
(2) $\delta(\mathsf{decs}(I)) = \{1, \ldots, \delta(I)\}$
(3) $\forall n \in \mathbb{N}\,.\, F \wedge \neg M \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$, *provided $\mathcal{C}$ is open or decomposed*

(4) $M \vee O(I)$ *is a d-DNNF, provided* $\mathcal{C}$ *is open or decomposed*
(5) $M \vee F \wedge O(I) \equiv F$
(6) $e > 0$ *iff (A)* $e = 2$*, (B)* $\mathcal{C}$ *is decomposed, (C)* $\mathcal{S}$ *contains components* $\mathcal{C}_G = (G, \mathsf{var}(G), d \cdot 1, e_g, J_G, N, \delta_G)^s$, $\mathcal{C}_H = (H, \mathsf{var}(H), d \cdot 2, e_H, J_H, O, \delta_H)^s$, *such that* $F|_I = G \wedge H$ *and* $\mathsf{var}(G) \cap \mathsf{var}(H) = \emptyset$
(7) If $e = 2$ *and* $\mathcal{S}$ *contains components* $\mathcal{C}_G = (G, \mathsf{var}(G), d \cdot 1, 0, J_G, N, \delta_G)^c$ *and* $\mathcal{C}_H = (H, \mathsf{var}(H), d \cdot 2, 0, J_H, O, \delta_H)^c$*, then* $F|_I \equiv I \wedge N \wedge O$
(8) If $\mathcal{C}$ *is closed, then* $\mathsf{decs}(I) = \emptyset$

Invariants (1) - (5) correspond to the ones in our previous work [34]. They say that decisions are ordered in ascending order with respect to their decision level and that every decision level contains a decision literal. They further ensure that literals propagated after backtracking upon finding a model are indeed implied, that no model is enumerated multiple times and that all models are found. Invariant (3) is only useful for open or decomposed components, since I remains unaltered when a component is closed. Invariant (4) only holds for closed components if $I(F) = \bot$. Invariants (6) and (7) are concerned with the properties of a parent component and its subcomponents (for the case $c = 2$), such as the definition of the component level. Since, given a trail I, $F|_I$ is decomposed into formulae G and H, we also have that $F|_I \equiv N \wedge O$, where $N \equiv G$ and $O \equiv H$. Finally, Inv. (8) says that the trail of a closed component contains no decision.

Lemma 1 (Soundness of the Initial Global State). *The initial global state* $\mathcal{S}_0 = \{(F, V, 0, 0, \varepsilon, \bot, \infty)^o\}$ *is sound.*

Proof. Due to $I = \varepsilon$ and $e = 0$ and since the (root) component is open, all invariants in Definition 1 are trivially met.

Theorem 1 (Soundness of ACD Rules). *The rules of* ACD *preserve soundness, i.e., they transform a sound global state into another sound global state.*

Proof. The proof is carried out by induction over the rule applications. We assume that prior to the application of a rule the invariants in Definition 1 are met and show that they also hold in the target state. The (parent) component in the original state is denoted by $\mathcal{C} = (F, V, d, e, I, M, \delta)^s$ and in the target state by $\mathcal{C}' = (F, V, d', e', I', M', \delta')^{s'}$. Its subcomponents, if there are any, are written $\mathcal{C}_G = (G, \mathsf{var}(G), d \cdot 1, e_G, J, N, \delta_G)^s$, $\mathcal{C}_H = (H, \mathsf{var}(H), d \cdot 2, e_H, K, O, \delta_H)^s$. Unit, Decide, BackTrue, and BackFalse: Apart from the additional elements V, d, e and the component state s, the rules are defined as in the former calculus [34]. The arguments given in the proof there apply here as well, and after applying rules Unit, Decide, BackTrue, or BackFalse, Inv. (1) - (5) hold. Notice that in the proof of Inv. (4), it suffices to replace "DSOP" by "d-DNNF", since the relevant property here is determinism. Since $e' = 0$, Inv. (6) and (7) do not apply. An open state is mapped to an open state, hence Inv. (8) holds.

CompTrue and CompFalse: Invariants (1) and (2) hold, since I remains unaffected. Since $\mathcal{C}'$ is closed, Inv. (3) and (4) are met. The proof that Inv. (5) holds is carried

out similarly to the proof of Proposition 1 in our previous work [34] for rules EndTrue and EndFalse, respectively. Since $e' = 0$ and $I' = I$, Inv. (6) - (8) hold.

Decompose: The parent component $\mathcal{C}$ remains unaltered except for $e' = 2$ and for its state, which becomes f. Both its subcomponents $\mathcal{C}_G$ and $\mathcal{C}_H$ are open, and we have $J_G = J_H = \varepsilon$ and $e_G = e_H = 0$. Therefore, Inv. (1) - (5) hold. Invariant (6) is satisfied by the definition of rule Decompose. Since $\mathcal{C}'$ is decomposed and $\mathcal{C}_G$ and $\mathcal{C}_H$ are open by definition, Inv. (7) and (8) hold as well.

ComposeBack: It suffices to show that the validity of the invariants for $\mathcal{C}'$ is preserved, since $\mathcal{C}_G$ and $\mathcal{C}_H$ do not occur in the target state. The most recent decision literal is flipped, similar to rule BackTrue. The same argument to the one given there applies, and Inv. (1) and (2) are satisfied. We need to show that $F \wedge \neg(M \vee (I \wedge N \wedge O)) \wedge \mathsf{decs}_{\leqslant n}(PK\ell) \models (PK\ell)_{\leqslant n}$ holds for all n. The decision levels of the literals in PK do not change, except for the one of ℓ, which is decremented from $e{+}1$ to e. The literal ℓ also stops from being a decision literal. Since $\delta(PK\ell) = e$, we can assume $n \leq e$. Furthermore, $F \wedge \neg(M \vee (I \wedge N \wedge O)) \wedge \mathsf{decs}_{\leqslant n}(PK\ell) \equiv (\neg I \wedge (F \wedge \neg M \wedge \mathsf{decs}_{\leqslant n}(I))) \vee (F \wedge \neg M \wedge \neg(N \wedge O) \wedge \mathsf{decs}_{\leqslant n}(I))$, since ℓ is not a decision literal in $PK\ell$ and $I_{\leqslant e} = PK$ and thus $I_{\leqslant n} = (PK)_{\leqslant n}$ by definition. By applying the induction hypothesis, we get $\neg I \wedge F \wedge \neg M \wedge \mathsf{decs}_{\leqslant n}(PK\ell) \models (PK)_{\leqslant n}$, and hence $F \wedge \neg(M \vee (I \wedge N \wedge O)) \wedge \mathsf{decs}_{\leqslant n}(PK\ell) \models (PK)_{\leqslant n}$. We still need to show that $F \wedge \neg(M \vee (I \wedge N \wedge O)) \wedge \mathsf{decs}_{\leqslant e}(PK\ell) \models \ell$, as $\delta(\ell) = e$ in $PK\ell$ after applying ComposeBack and thus ℓ disappears from the proof obligation for $n < e$. Notice that $F \wedge \neg D \models I$ using again the induction hypothesis for $n = e + 1$. This gives us $F \wedge \neg\mathsf{decs}_{\leqslant e}(PK) \wedge \neg\ell \models I$ and thus $F \wedge \neg\mathsf{decs}_{\leqslant e}(PK) \wedge \neg I \models \ell$ by conditional contraposition, and Inv. (3) holds.

For proving that Inv. (4) holds, we consider two cases: (A) $I \wedge N \wedge O \neq \bot$, i.e., there exists an extension of I which satisfies F, and (B) $I \wedge N \wedge O = \bot$, i.e., all extensions of I falsify F. For both cases, we know that $I \vee O(I)$ is a d-DNNF.

(A) We need to show that $M \vee (I \wedge N \wedge O) \vee O(PK\ell)$ is a d-DNNF. Due to $\delta(I) = e + 1$, we have $O(I) = I \vee R_{\leqslant e+1}(I) = I \vee R_{\leqslant e}(I) \vee R_{=e+1}(I)$. The pending search space of $PK\ell$ is given by $O(PK\ell) = PK\ell \vee R_{\leqslant e}(PK\ell)$. But $PK = I_{\leqslant e}$ and $PK\ell = I_{\leqslant e}\ell = R_{=e+1}(I)$, since $\neg\ell \in \mathsf{decs}(I)$ and $\delta(\neg\ell) = e + 1$. Furthermore, $R_{\leqslant e}(PK\ell) = R_{\leqslant e}(PK)$, since $\ell \notin \mathsf{decs}(PK\ell)$ and $\delta(\ell) = e$, hence $R_{\leqslant e}(PK\ell) = R_{\leqslant e}(I)$. We have $O(PK\ell) = R_{=e+1}(I) \vee R_{\leqslant e}(I)$, hence $O(PK\ell) \vee I = O(I)$ and $(M \vee I) \vee O(PK\ell) = M \vee O(I)$, which is a DSOP and hence a d-DNNF. Now I, N, and O are defined over pairwise disjoint sets of variables by construction, i.e., $I \wedge N \wedge O$ is decomposable, and $M \vee (I \wedge N \wedge O) \vee O(PK\ell)$ is a d-DNNF.

(B) We need to show that $M \vee O(PK\ell)$ is a d-DNNF. As just shown, $O(PK\ell) \vee I = O(I)$. Now $M \vee O(PK\ell) = M \vee R_{\leqslant e+1}(I)$. Recalling that $R_{\leqslant e+1}(I)$ is equal to $O(I)$ without I and $M \vee O(I)$ is a d-DNNF by the premise, $M \vee O(PK\ell)$ is a d-DNNF as well. Therefore, Inv. (4) holds.

For the proof of the validity of Inv. (5), given $M \vee F \wedge O(I) \equiv F$, the same two cases are relevant: (A) $I \wedge N \wedge O \neq \bot$ and (B) $I \wedge N \wedge O = \bot$.

(A) We have to show that $M \vee (I \wedge N \wedge O) \vee (F \wedge O(PK\ell)) \equiv F$. From $O(PK\ell) \vee I = O(I)$ we get $M \vee (F \wedge O(I)) = M \vee (F \wedge (O(PK\ell)) \vee I) =$

$M \vee (F \wedge O(P K \ell)) \vee (F \wedge I) \equiv F$. But $F \wedge I \equiv I \wedge N \wedge O$. Therefore $M \vee (F \wedge O(I)) \equiv M \vee (F \wedge O(P K \ell)) \vee (I \wedge N \wedge O) = M \vee (I \wedge N \wedge O) \vee (F \wedge O(P K \ell)) \equiv F$.

(B) We must show that $M \vee (F \wedge O(P K \ell)) \equiv F$. Similarly to (A) we have $M \vee (F \wedge O(I)) \equiv M \vee (F \wedge O(P K \ell)) \vee (F \wedge I) \equiv M \vee (F \wedge O(P K \ell)) \equiv F$, due to $F \wedge I \equiv F$. Therefore, Inv. (5) holds after applying rule ComposeBack. We have $e' = 0$, and $\mathcal{C}'$ is open, hence Inv. (6) - (8) trivially hold.

ComposeEnd: It suffices to show that after applying rule ComposeBack the invariants are met by $\mathcal{C}'$, since its subcomponent states $\mathcal{C}_G$ and $\mathcal{C}_H$ do not occur in the target state anymore. Due to $I' = I$ and $\mathsf{decs}(I) = \emptyset$ and since $\mathcal{C}'$ is closed, Inv. (1) - (4) trivially hold.

For proving that invariant (5) holds after applying rule ComposeEnd, i.e., that $M \vee (I \wedge N \wedge O) \vee (F \wedge O(I)) \equiv F$, the same two cases need to be distinguished: (A) $I \wedge N \wedge O \neq \bot$ and (B) $I \wedge N \wedge O = \bot$.

(A) From $\mathsf{decs}(I) = \emptyset$, we get $O(I) = I$ and $F \wedge O(I) = F \wedge I$. Recalling that $F \wedge I \equiv I \wedge N \wedge O$, we obtain $M \vee (I \wedge N \wedge O) \vee (F \wedge O(I)) \equiv M \vee (F \wedge O(I)) \equiv F$ by the premise.

(B) We have $M \vee (I \wedge N \wedge O) \vee (F \wedge O(I)) = M \vee (F \wedge O(I)) \equiv F$ by the premise, and Inv. (5) holds after executing rule ComposeEnd. Invariants (6) - (8) trivially hold, due to $e' = 0$ and $I' = I$ and hence $\mathsf{decs}(I') = \emptyset$.

Corollary 1 (Soundness of ACD Run). ACD *starting with an initial global state is sound.*

Proof. The initial state is sound by Lemma 1, and all rule applications lead to a sound state according to Theorem 1.

Lemma 2 (Correctness of Closed Component State). *For any closed component* $(F, V, d, 0, I, M, \delta)^c$ *it holds that* $M \equiv F$.

Proof. Follows from Theorem 1, proof of Inv. (5) for rules CompTrue, CompFalse, and ComposeEnd, which are the only rules closing a component.

Theorem 2 (Correctness of Final Global State). *In the final global state* $\mathcal{S}_n = \{(F, V, d, 0, I, M, \delta)^c\}$ *of* ACD, $M \equiv F$ *holds.*

Proof. Correctness of the closed root component follows from Lemma 2. We need to show that the final global state contains exactly the closed root component. The initial global state consists of the open root component. Additional components are created exclusively by rule Decompose, and a parent component state can only be closed by rule ComposeEnd, which also removes its subcomponents from the global state. Hence the root component can only be closed if it has no subcomponents. But since the initial global state contains exclusively the root component, the final global state contains only the closed root component.

Theorem 3 (Progress). ACD *always makes progress.*

Unit:
$\mathcal{S}' \uplus \{(d, [l_1, \ldots, l_k, 2, 2, \ldots, 2], o)\}$
$\succ_{\text{ACD}}$
$\mathcal{S}' \uplus \{(d, [l_1, \ldots, l_k, 0, 2, \ldots, 2], o)\}$

Decide:
$\mathcal{S}' \uplus \{(d, [l_1, \ldots, l_k, 2, 2, \ldots, 2], o)\}$
$\succ_{\text{ACD}}$
$\mathcal{S}' \uplus \{(d, [l_1, \ldots, l_k, 1, 2, \ldots, 2], o)\}$

BackTrue:
$\mathcal{S}' \uplus \{(d, [l_1, \ldots, l_k, 1, l_{k+2}, \ldots, l_{|V|}], o)\}$
$\succ_{\text{ACD}}$
$\mathcal{S}' \uplus \{(d, [l_1, \ldots, l_k, 0, l'_{k+2}, \ldots, l'_{|V|}], o)\}$

BackFalse:
$\mathcal{S}' \uplus \{(d, [l_1, \ldots, l_k, 1, l_{k+2}, \ldots, l_{|V|}], o)\}$
$\succ_{\text{ACD}}$
$\mathcal{S}' \uplus \{(d, [l_1, \ldots, l_k, 0, l'_{k+2}, \ldots, l'_{|V|}], o)\}$

CompTrue:
$\mathcal{S}' \uplus \{(d, t, o)\} \succ_{\text{ACD}} \mathcal{S}' \uplus \{(d, t, c)\}$

CompFalse:
$\mathcal{S}' \uplus \{(d, t, o)\} \succ_{\text{ACD}} \mathcal{S}' \uplus \{(d, t, c)\}$

Decompose:
$\mathcal{S}' \uplus \{(d, t, o)\} \succ_{\text{ACD}} \mathcal{S}' \uplus \{(d, t, f), (d \cdot 1, [2, \ldots, 2], o), (d \cdot 2, [2, \ldots, 2], o)\}$

ComposeBack:
$\mathcal{S}' \uplus \{(d, [l_1, \ldots, l_k, 1, l_{k+2}, \ldots, l_{|V|}], f), (d \cdot 1, t_1, c), (d \cdot 2, t_2, c)\}$
$\succ_{\text{ACD}}$
$\mathcal{S}' \uplus \{(d, [l_1, \ldots, l_k, 0, l'_{k+2}, \ldots, l'_{|V|}, o])\}$

ComposeEnd:
$\mathcal{S}' \uplus \{(d, t, f), (d \cdot 1, t_1, c), (d \cdot 2, t_2, c)\} \succ_{\text{ACD}} \mathcal{S}' \uplus \{(d, t, c)\}$

Fig. 4. Rule applications lead to smaller global states.

Proof. The proof is conducted by induction over the rules. We show that as long as the root component is not closed, a rule is applicable. For the case $\mathcal{S} \uplus \{\mathcal{C}\}$, where $\mathcal{C} = (F, V, d, 0, I, M, \delta)^o$ has no subcomponents, the proof is identical to the one showing progress in our previous work [34] replacing EndTrue with CompTrue and EndFalse with CompFalse, and by checking whether the preconditions for rule Decompose are met if rule Unit is not applicable and before taking a decision. Now let the global state be given by $\mathcal{S} \uplus \{\mathcal{C}\}$ where $\mathcal{C} = (F, V, d, 2, I, M, \delta)^f$ is decomposed. Due to Inv. (6), $\mathcal{S}$ contains $\mathcal{C}_G = (G, \mathsf{var}(G), d \cdot 1, e_G, J_G, N, \delta_G)^s$ and $\mathcal{C}_H = (H, \mathsf{var}(H), d \cdot 2, e_H, J_H, O, \delta_H)^s$ such that $F|_I = G \wedge H$ and $\mathsf{var}(G) \cap \mathsf{var}(H) = \emptyset$. Assume $s = c$ for both $\mathcal{C}_G$ and $\mathcal{C}_H$. If $\mathsf{decs}(I) = \emptyset$, rule ComposeEnd is applicable. Otherwise, similarly to rule BackTrue, we can show that all preconditions of rule ComposeBack are met. If instead $s \in \{f, o\}$ for at least one of $\mathcal{C}_G$ and $\mathcal{C}_H$, the non-closed component(s) are processed further, and as soon as both $\mathcal{C}_G$ and $\mathcal{C}_H$ are closed, rule ComposeEnd or ComposeBack can be applied. This proves that ACD always makes progress.

Theorem 4 (Termination). ACD *always terminates.*

Proof. We need to show that no infinite sequence of rule applications can happen. To this end, we define a strict, well-founded ordering $\succ_{\text{ACD}}$ on the global states and show that $\mathcal{S} \rightsquigarrow_{\mathsf{R}} \mathcal{T}$ implies $\mathcal{S} \succ_{\text{ACD}} \mathcal{T}$ for all $\mathcal{S}, \mathcal{T} \in S$ and rules R in ACD. Global states are sets of components, and $\succ_{\text{ACD}}$ is the multiset extension of a component ordering $\succ_{\mathsf{c}} = (\succ_{\mathsf{cl}}, \succ_{\mathsf{tr}}, \succ_{\mathsf{cs}})$, where $\succ_{\mathsf{cl}}$, $\succ_{\mathsf{tr}}$, and $\succ_{\mathsf{cs}}$ are orderings on component levels, trails, and component states, respectively. We want to compare trails defined over the same set of variables V, and to this end we represent them

DecomposeG:	$\mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^o\} \rightsquigarrow_{\mathsf{DecomposeG}}$
	$\mathcal{S} \uplus \{(F, V, d, n, I, M, \delta)^f, ((G_i, U_i, d \cdot i, 0, \varepsilon, \bot, \infty)^o)_{i=1}^n, \}$ if $F\|_I \neq \top$ and
	$\bot \notin F\|_I$ and $\mathsf{units}(F\|_I) = \emptyset$ and $\bigwedge_{i=1}^n G_i \stackrel{\text{def}}{=} F\|_I$ and $n \geq 2$ and
	$U_i \stackrel{\text{def}}{=} \mathsf{var}(G_i)$ and $U_i \cap U_j = \emptyset$ for $1 \leq i, j \leq n$ and $i \neq j$
ComposeBackG:	$\mathcal{S} \uplus \{(F, V, d, n, I, M, \delta)^f, ((G_i, U_i, d \cdot i, 0, J_i, N_i, \delta_i)^c)_{i=1}^n\} \rightsquigarrow_{\mathsf{ComposeBackG}}$
	$\mathcal{S} \uplus \{(F, V, d, 0, PK\ell, M \vee I \wedge N, \delta[L \mapsto \infty][\ell \mapsto e])^o\}$ if $PQ \stackrel{\text{def}}{=} I$ and
	$D \stackrel{\text{def}}{=} \neg\mathsf{decs}(I)$ and $e + 1 \stackrel{\text{def}}{=} \delta(D \setminus \{\ell\}) = \delta(I)$ and $\ell \in D$ and
	$e = \delta(D \setminus \{\ell\}) = \delta(P)$ and $K \stackrel{\text{def}}{=} Q_{\leqslant e}$ and $L \stackrel{\text{def}}{=} Q_{>e}$ and $N \stackrel{\text{def}}{=} \bigwedge_{i=1}^n N_i$
ComposeEndG:	$\mathcal{S} \uplus \{(F, V, d, n, I, M, \delta)^f, ((\top, U_i, d \cdot i, 0, J_i, N_i, \delta_i)^c)_{i=1}^n\} \rightsquigarrow_{\mathsf{ComposeEndG}}$
	$\mathcal{S} \uplus \{(F, V, d, 0, I, M \vee I \wedge N, \delta[I \mapsto \delta])^c\}$ if $\mathsf{decs}(I) = \emptyset$ and $N \stackrel{\text{def}}{=} \bigwedge_{i=1}^n N_i$

Fig. 5. Generalized transition rules.

as lists over $\{0, 1, 2\}$. A trail $I = \ell_1 \ldots \ell_k$ defined over V, where $k \leq |V|$, is represented as $[l_1, \ldots, l_k, 2, \ldots, 2]$, where $l_i = 0$ if ℓ_i is a propagation literal and $l_i = 1$ if ℓ_i is a decision literal. The last $|V| - m$ positions with value 2 represent the unassigned variables. Trails defined over the same variable set are encoded into lists of the same length. This representation induces a lexicographic order $>_{\mathsf{lex}}$ on trails, and we define $\succ_{\mathsf{tr}}$ as the restriction of $>_{\mathsf{lex}}$ to $\{[l_1, \ldots, l_{|V|}] \mid l_i \in \{0, 1, 2\}$ for $1 \leq i \leq |V|\}$, i.e., we have $t_1 \succ_{\mathsf{tr}} t_2$ if $t_1 >_{\mathsf{lex}} t_2$. The ordering $\succ_{\mathsf{tr}}$ is well-founded, its minimal element is $[0, \ldots, 0]$. The component state takes values in $\{o, f, c\}$, and we define $\succ_{\mathsf{cs}}$ as $>_{\mathsf{lex}}$, i.e., $s_1 \succ_{\mathsf{cs}} s_2$ if $s_1 >_{\mathsf{lex}} s_2$. The minimal element of $\succ_{\mathsf{cs}}$ is c, hence $\succ_{\mathsf{cs}}$ is well-founded. Given two component levels d_1 and d_2, we define $d_1 \succ_{\mathsf{cl}} d_2$ if $\mathsf{length}(d_1) < \mathsf{length}(d_2)$. This may seem counterintuitive but is needed to ensure that the execution of rule Decompose results in a smaller state, since both the component state and the trail of the new subcomponents are of higher order than those of their parent. To see that $\succ_{\mathsf{cs}}$ is well-founded, recall that we consider finite variable sets. Their size provides an upper limit on the length of the component level representation and a minimal element of $\succ_{\mathsf{cs}}$.

Now we define the component ordering $\succ_{\mathsf{c}} = (\succ_{\mathsf{cl}}, \succ_{\mathsf{tr}}, \succ_{\mathsf{cs}})$. Let two components be $\mathcal{C}_1 = (d_1, t_1, s_1)$ and $\mathcal{C}_2 = (d_2, t_2, s_2)$. We have $\mathcal{C}_1 \succ_{\mathsf{c}} \mathcal{C}_2$ if $\mathcal{C}_1 \neq \mathcal{C}_2$ and $d_1 \succ_{\mathsf{cl}} d_2$ or $d_1 = d_2$ and either $t_1 \succ_{\mathsf{tr}} t_2$ or $t_1 = t_2$ and $s_1 \succ_{\mathsf{cs}} s_2$. Clearly $\succ_{\mathsf{c}}$ is well-founded, since $\succ_{\mathsf{tr}}$, $\succ_{\mathsf{cs}}$, and $\succ_{\mathsf{cl}}$ are well-founded. For two global states $\mathcal{S}$ and $\mathcal{T}$, we have $\mathcal{S} \succ_{\mathrm{ACD}} \mathcal{T}$ if $\mathcal{S} \neq \mathcal{T}$ and for each component $\mathcal{C}$ such that $\mathcal{C}$ is larger in $\mathcal{T}$ than in $\mathcal{S}$ with respect to $\succ_{\mathsf{c}}$, $\mathcal{S}$ contains a component $\mathcal{C}'$ that is larger in $\mathcal{S}$ than in $\mathcal{T}$. Since $\succ_{\mathsf{c}}$ is well-founded, also $\succ_{\mathrm{ACD}}$ is well-founded. Figure 4 shows that each rule application leads to a smaller global state, concluding our proof.

6 Generalization

The generalized rules are listed in Fig. 5. In our generalized framework, we have $F|_I = \bigwedge_{i=1}^n G_i$, and $\mathsf{var}(G_i) \cap \mathsf{var}(G_j) = \emptyset$ for $i, j \in \{1, \ldots, n\}$ and $i \neq j$ (rule DecomposeG). Similarly to their equivalents in ACD, rules ComposeBackG and ComposeEndG are applicable if all subcomponents are closed.

7 Discussion

We have presented Abstract CNF2dDNNF, or ACD for short, a formal framework for compiling a formula in CNF into d-DNNF combining CDCL-based model enumeration with chronological backtracking [34] and dynamic component analysis [4]. Conflict-driven clause learning enables our framework to escape regions without solution early, and chronological backtracking prevents multiple model enumeration without the need for remembering already found models using blocking clauses, which slow down unit propagation. However, the absence of blocking clauses also prevents the use of restarts. If exclusively the rules Unit, Decide, BackTrue, BackFalse, CompTrue, and CompFalse are used, a DSOP representation of F is computed. Unit propagation is prioritized due to its potential to reduce the number of decisions and thus of right branches to be explored. Favoring decompositions over decisions may also shrink a larger part of the search space. Our framework lays the theoretical foundation for practical All-SAT and #SAT solving based on chronological CDCL. Any implementation which can be modeled by ACD exhibits its properties, in particular its correctness, which has been established in a formal proof.

Comparison with Available Tools. There exist other knowledge compilers addressing d-DNNFs. We want to mention c2d [20], Dsharp [37], and D4 [30], which also execute an exhaustive search and conflict analysis. However, our approach differs conceptually from these tools in several ways. The most prominent ones are the use of CDCL with chronological backtracking [33,38] instead of CDCL with non-chronological backtracking and the way the d-DNNF is created. Our method generates DSOP representations of formulae which can not be decomposed further by an exhaustive (partial) model enumeration and then combines the result, while the tools mentioned above generate the d-DNNF by recording the execution trace as a graph [26,27]. As ACD, both D4 and Dsharp adopt a dynamic decomposition strategy, while c2d constructs a decomposition tree which it then uses for for component analysis.

Future Research Directions. We plan to implement a proof of concept of our calculus in order to compare the size of the returned d-DNNF with the ones obtained by c2d, D4, and Dsharp. For dynamic component analysis, one could follow the algorithm implemented in COMPSAT [6], while dual reasoning [32] and logical entailment [35] enable the detection of short partial models. This is particularly interesting in tasks where the length of the d-DNNF is crucial. Dual reasoning has shown to be almost competitive on CNFs if the search space is small, we therefore expect that component analysis boosts its performance. The major challenge posed by the second approach lies in an efficient implementation of the oracle calls required by the entailment checks. It would be interesting to investigate the impact of dynamic component analysis on a recent implementation [46] of model enumeration by chronological CDCL [34]. Cache structures, being an inherent part of modern knowledge compilers and #SAT solvers [11,16,19,20,30,31,37,41,42,47,49] due to their positive impact on solver efficiency [1], should be added to any implementation of our framework. Finally,

an important research topic is that of optimizing the encoding of a formula making best use of component analysis [14]. Related to this question is whether formulae stemming from practical applications are decomposable in general.

Acknowledgements. My thanks go to Armin Biere for a fruitful discussion when I got stuck in a first, very raw version of the proof, and to Martin Bromberger for his input enhancing it.

References

1. Bacchus, F., Dalmao, S., Pitassi, T.: DPLL with caching: a new algorithm for #SAT and Bayesian inference. Electron. Colloquium Comput. Complex. **TR03-003** (2003)
2. Barrett, A.: From hybrid systems to universal plans via domain compilation. In: ICAPS, pp. 44–51. AAAI (2004)
3. Barrett, A.: Model compilation for real-time planning and diagnosis with feedback. In: IJCAI, pp. 1195–1200. Professional Book Center (2005)
4. Bayardo Jr., R., Pehoushek, J.: Counting models using connected components. In: AAAI/IAAI, pp. 157–162. AAAI Press/The MIT Press (2000)
5. Bernasconi, A., Ciriani, V., Luccio, F., Pagli, L.: Compact DSOP and partial DSOP forms. Theory Comput. Syst. **53**(4), 583–608 (2013)
6. Biere, A., Sinz, C.: Decomposing SAT problems into connected components. J. Satisf. Boolean Model. Comput. **2**(1–4), 201–208 (2006)
7. Bollig, B., Buttkus, M.: On limitations of structured (deterministic) DNNFs. Theory Comput. Syst. **64**(5), 799–825 (2020)
8. Bollig, B., Farenholtz, M.: On the relation between structured d-DNNFs and SDDs. Theory Comput. Syst. **65**(2), 274–295 (2021)
9. Bova, S., Capelli, F., Mengel, S., Slivovsky, F.: On compiling CNFs into structured deterministic DNNFs. In: Heule, M., Weaver, S. (eds.) SAT 2015. LNCS, vol. 9340, pp. 199–214. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24318-4_15
10. Bova, S., Capelli, F., Mengel, S., Slivovsky, F.: Knowledge compilation meets communication complexity. In: IJCAI, pp. 1008–1014. IJCAI/AAAI Press (2016)
11. Burchard, J., Schubert, T., Becker, B.: Laissez-faire caching for parallel #SAT solving. In: Heule, M., Weaver, S. (eds.) SAT 2015. LNCS, vol. 9340, pp. 46–61. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24318-4_5
12. Burchard, J., Schubert, T., Becker, B.: Distributed parallel #SAT solving. In: CLUSTER, pp. 326–335. IEEE Computer Society (2016)
13. Cadoli, M., Donini, F.M.: A survey on knowledge compilation. AI Commun. **10**(3–4), 137–150 (1997)
14. Chavira, M., Darwiche, A.: Encoding CNFs to empower component analysis. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 61–74. Springer, Heidelberg (2006). https://doi.org/10.1007/11814948_9
15. Chavira, M., Darwiche, A., Jaeger, M.: Compiling relational Bayesian networks for exact inference. Int. J. Approx. Reason. **42**(1–2), 4–20 (2006)
16. Chu, G., Harwood, A., Stuckey, P.J.: Cache conscious data structures for Boolean satisfiability solvers. J. Satisf. Boolean Model. Comput. **6**(1–3), 99–120 (2009)
17. Darwiche, A.: Compiling knowledge into decomposable negation normal form. In: IJCAI, pp. 284–289. Morgan Kaufmann (1999)

18. Darwiche, A.: Decomposable negation normal norm. J. ACM **48**(4), 608–647 (2001)
19. Darwiche, A.: On the tractable counting of theory models and its application to truth maintenance and belief revision. J. Appl. Non Class. Logics **11**(1–2), 11–34 (2001)
20. Darwiche, A.: New advances in compiling CNF into decomposable negation normal form. In: ECAI, pp. 328–332. IOS Press (2004)
21. Darwiche, A.: SDD: a new canonical representation of propositional knowledge bases. In: IJCAI, pp. 819–826. IJCAI/AAAI (2011)
22. Darwiche, A., Marquis, P.: A knowledge compilation map. J. Artif. Intell. Res. **17**, 229–264 (2002)
23. Davis, M., Logemann, G., Loveland, D.W.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (1962)
24. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**(3), 201–215 (1960)
25. Fargier, H., Mengin, J.: A knowledge compilation map for conditional preference statements-based languages. In: AAMAS, pp. 492–500. ACM (2021)
26. Huang, J., Darwiche, A.: DPLL with a trace: From SAT to knowledge compilation. In: IJCAI, pp. 156–162. Professional Book Center (2005)
27. Huang, J., Darwiche, A.: The language of search. J. Artif. Intell. Res. **29**, 191–219 (2007)
28. Huang, X., Izza, Y., Ignatiev, A., Cooper, M.C., Asher, N., Marques-Silva, J.: Tractable explanations for d-DNNF classifiers. In: AAAI, pp. 5719–5728. AAAI Press (2022)
29. Koriche, F., Lagniez, J., Marquis, P., Thomas, S.: Knowledge compilation for model counting: affine decision trees. In: IJCAI, pp. 947–953. IJCAI/AAAI (2013)
30. Lagniez, J., Marquis, P.: An improved Decision-DNNF compiler. In: IJCAI, pp. 667–673. ijcai.org (2017)
31. Lagniez, J., Marquis, P., Szczepanski, N.: DMC: a distributed model counter. In: IJCAI, pp. 1331–1338. ijcai.org (2018)
32. Möhle, S., Biere, A.: Dualizing projected model counting. In: ICTAI, pp. 702–709. IEEE (2018)
33. Möhle, S., Biere, A.: Backing backtracking. In: Janota, M., Lynce, I. (eds.) SAT 2019. LNCS, vol. 11628, pp. 250–266. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24258-9_18
34. 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)
35. Möhle, S., Sebastiani, R., Biere, A.: Four flavors of entailment. In: Pulina, L., Seidl, M. (eds.) SAT 2020. LNCS, vol. 12178, pp. 62–71. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51825-7_5
36. 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)
37. Muise, C., McIlraith, S.A., Beck, J.C., Hsu, E.I.: Dsharp: fast d-DNNF compilation with sharpSAT. In: Kosseim, L., Inkpen, D. (eds.) AI 2012. LNCS (LNAI), vol. 7310, pp. 356–361. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30353-1_36
38. Nadel, A., Ryvchin, V.: Chronological backtracking. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) SAT 2018. LNCS, vol. 10929, pp. 111–121. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94144-8_7

39. Palacios, H., Bonet, B., Darwiche, A., Geffner, H.: Pruning conformant plans by counting models on compiled d-DNNF representations. In: ICAPS, pp. 141–150. AAAI (2005)
40. Pipatsrisawat, K., Darwiche, A.: A new d-DNNF-based bound computation algorithm for functional E-MAJSAT. In: IJCAI, pp. 590–595 (2009)
41. Sang, T., Bacchus, F., Beame, P., Kautz, H.A., Pitassi, T.: Combining component caching and clause learning for effective model counting. In: SAT (2004)
42. Sharma, S., Roy, S., Soos, M., Meel, K.S.: GANAK: a scalable probabilistic exact model counter. In: IJCAI, pp. 1169–1176. ijcai.org (2019)
43. Siddiqi, S.A., Huang, J.: Probabilistic sequential diagnosis by compilation. In: ISAIM (2008)
44. Marques-Silva, J.M., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: ICCAD, pp. 220–227. IEEE Computer Society/ACM (1996)
45. Marques-Silva, J.M., Sakallah, K.A.: GRASP: a search algorithm for propositional satisfiability. IEEE Trans. Comput. **48**(5), 506–521 (1999)
46. Spallitta, G., Sebastiani, R., Biere, A.: Enumerating disjoint partial models without blocking clauses. CoRR abs/2306.00461 (2023)
47. Thurley, M.: sharpSAT – counting models with advanced component caching and implicit BCP. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 424–429. Springer, Heidelberg (2006). https://doi.org/10.1007/11814948_38
48. de Uña, D., Gange, G., Schachte, P., Stuckey, P.J.: Compiling CP subproblems to MDDs and d-DNNFs. Constraints An Int. J. **24**(1), 56–93 (2019)
49. Zhang, L., Malik, S.: Cache performance of SAT solvers: a case study for efficient implementation of algorithms. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 287–298. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24605-3_22