

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

Sibylle Möhle^[0000-0001-7883-7749]

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 means of it. We provide a rigorous correctness proof and a detailed example. The main conceptual differences to currently available tools targeting d-DNNFs are discussed and future research directions presented. The aim of this work is to introduce a novel method for d-DNNF compilation with focus on model computation. 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 reasoner¹.

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

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

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 significantly reduce the amount of work per computation. 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 $V_1 \cap V_2 = \emptyset$, we can compute d-DNNF representations of F_1 and F_2 independently and conjoin them 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 the disjuncts are pairwise contradicting, satisfies *determinism* [22], and a deterministic DNNF formula is in d-DNNF. Determinism is also met by the language disjoint sum-of-products (DSOP), which is a disjunction of pairwise contradicting conjunctions of literals, and which is relevant in circuit design [5]. In previous work [34], an approach for translating a CNF formula into DSOP based on CDCL with chronological backtracking has been introduced. The reason 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 use of blocking clauses commonly used in model enumeration. They rule out already found models but also slow down the solver. Enhancing this approach by component-based reasoning enables us to compute a d-DNNF representation of a CNF formula. Reconsider our previous example, and suppose we obtained $\text{dsop}(F_1) = a \vee (\neg a \wedge b)$ and $\text{dsop}(F_2) = c \vee (\neg c \wedge d)$. Now $F \equiv F_1 \wedge F_2$, hence $F \equiv \text{dsop}(F_1) \wedge \text{dsop}(F_2) = (a \vee (\neg a \wedge b)) \wedge (c \vee (\neg c \wedge d))$, 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 rigorous 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 its means is sound as well. This comprises optimizations and implementation details, such as caches. ABSTRACT CNF2DDNNF combines component-based reasoning and CNF-to-DSOP compilation based on conflict-driven clause learn-

ing (CDCL) with chronological backtracking. Disjunctions with contradicting 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.

2 Preliminaries

Let V be a set of propositional variables defined over the set of Boolean constants \perp (false) and \top (true) denoted by $\mathbb{B} = \{\perp, \top\}$. A *literal* is either a variable $v \in V$ or its negation $\neg v$. We refer to the variable of a literal ℓ by $\text{var}(\ell)$ 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 contradicting *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 contradicting. We might interpret formulae as sets of clauses and cubes and clauses as sets of literals and write $C \in F$ 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 \perp .

A *total variable assignment* is a mapping $\sigma: V \mapsto \mathbb{B}$, 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 by writing $\ell \in I$ 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 $\text{var}(I)$. Trails and literals can be concatenated, written IJ and $I\ell$, given $\text{var}(I) \cap \text{var}(J) = \emptyset$ and $\text{var}(I) \cap \text{var}(\ell) = \emptyset$. 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 $\text{units}(F)$ ($\text{units}(F|_I)$) we denote the set of unit literals in F ($F|_I$). Similarly, $\text{decs}(I)$ denotes the set of decision literals on I . By writing $\ell \in \text{decs}(I)$ ($\ell \in \text{units}(F)$, $\ell \in \text{units}(F|_I)$), we refer to a decision literal ℓ on I (unit literal in F , $F|_I$). A trail I *falsifies* F , if $I(F) \equiv \perp$, i.e., $F|_I = \perp$. 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 $\text{var}(I) = V$, 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* $\delta: V \mapsto \mathbb{N} \cup \{\infty\}$ returns the decision level of a variable $v \in V$. If v is unassigned,

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

```

input : CNF  $F, V = \text{var}(F), I = \varepsilon, M = \perp$ 
output : d-DNNF  $M \equiv F$ 
1 Loop
2    $I \leftarrow \text{PropagateUnits}()$ 
3   if conflict occurs then
4     if conflict level = 0 then return  $M$ 
5     else  $\text{AnalyzeConflict}()$ 
6   else if  $I(F) = \top$  then
7      $M \leftarrow M \vee I$ 
8     if there are no decisions on  $I$  then return  $M$ 
9     else  $\text{BacktrackChrono}()$ 
10  else if  $F|_I$  can be decomposed into  $G$  and  $H$  then
11     $M \leftarrow M \vee I \wedge \text{CNF2dDNNF}(G, \text{var}(G), \varepsilon, \perp) \wedge \text{CNF2dDNNF}(H, \text{var}(H), \varepsilon, \perp)$ 
12    if there are no decisions on  $I$  then return  $M$ 
13    else  $\text{BacktrackChrono}()$ 
14  else Decide

```

$\delta(v) = \infty$, 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 $\delta(\ell) = \delta(\text{var}(\ell))$, $\delta(C) = \max\{\delta(\ell) \mid \ell \in C\}$ for $C \neq \perp$ and $\delta(I) = \max\{\delta(\ell) \mid \ell \in I\}$ for $I \neq \varepsilon$ extending this notation to sets of literals. Finally, we define $\delta(\perp) = \delta(\varepsilon) = \infty$. 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 [44,45,36], in *chronological CDCL* [33,38] literals may not be ordered on the trail in ascending order with respect to their decision level. We write $I_{\leq n}$ ($I_{< n}$, $I_{=n}$) 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 $R(I) = \bigvee_{\ell \in \text{decs}(I)} R_{=\delta(\ell)}(I)$ and $R_{=\delta(\ell)}(I) = \neg \ell \wedge I_{< \delta(\ell)}$ for $\ell \in \text{decs}(I)$. Consider $I = ab^d cde^d f$, $O(I) = (a \wedge b \wedge c \wedge d \wedge e \wedge f) \vee (\neg b \wedge a) \vee (\neg e \wedge a \wedge b \wedge c \wedge d)$. Similarly, the *pending models* of F are the satisfying assignments not yet found, $F \wedge O(I)$.

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 pseudocode. Lines 1–9 and 14 describe CDCL-based model enumeration with chronological backtracking [34], while lines 10–13 capture component analysis.

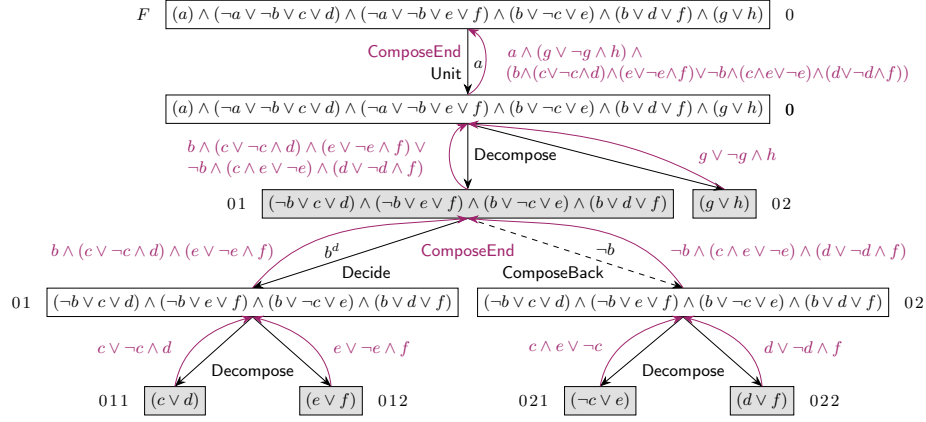


Fig. 1. Component structure of F created by ABSTRACT CNF2DDNNF.

Now assume unit propagation has been carried out until completion, no conflict has occurred and there are still unassigned variables (line 10). If $F|_I$ can be decomposed into two formulae G and H , we call CNF2dDNNF recursively on G and H , conjoin the outcomes of these computations with I and add the result to M (line 11). If I contains no decisions, the search space has been explored exhaustively, otherwise chronological backtracking occurs (lines 12–13). The working of our approach is shown by an example.

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 and component levels for now, they are intended to clarify the working of our calculus which is presented in Sec. 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). But before this result can be used further, the subcomponent at component level 01 needs to be pro-

cessed. Its formula is $F' = (\neg b \vee c \vee d) \wedge (\neg b \vee e \vee f) \wedge (b \vee \neg c \vee e) \wedge (b \vee d \vee f)$. It neither contains a unit nor can it be decomposed, hence we take a decision, let's say, b^d . Now $F'|_b = (c \vee d) \wedge (e \vee f)$, 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 F' was decomposed, a decision was taken, and we backtrack combining the results of its subcomponents (**ComposeBack**). We have $F'|_{\neg b} = (\neg c \vee e) \wedge (d \vee f)$ giving rise to two components with component level 021 and 022, 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 (**ComposeEnd**).

4 Calculus

Due to its recursive nature, combining the results computed for subcomponents in CNF2dDNNF is straightforward. In our non-recursive formalization, 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 elements of \mathbb{N}^+ as follows. Suppose a component is assigned level d and assume it is decomposed into two subformulae. Its subcomponents are assigned component level $d \cdot 1$ and $d \cdot 2$, respectively, and the level of their parent is given by the substring consisting of all but the last element of their level. The *root component* holds the input formula, it has no parent and component level 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 its subcomponents are closed but their results are not yet combined. Components which are neither closed nor decomposed are *open*.² Closed components may be discarded as soon as their results are combined, and the computation stops as soon as the root component is closed. With these remarks, we are now ready to present our calculus.

We describe our algorithm in terms of a state transition system **ABSTRACT CNF2dDNNF** over a set of global states S , a transition relation $\rightsquigarrow \subseteq S \times S$ and an initial global state S_0 . A *global state* is a set of components. A *component* \mathcal{C} is described as a seven-tuple $(F, V, d, c, I, M, \delta)^s$, 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 variables. The third element d denotes the component level of \mathcal{C} . If $d \neq 0$, then $d \in \{l \cdot 1, l \cdot 2\}$, where l is the component level of the parent component of \mathcal{C} . 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 c , 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 $(F, V, d, 0, \varepsilon, \perp, \infty)^o$ and closed after its computation has terminated, i.e.,

² The differentiation between open and decomposed components is purely technical and needed for the termination proof in Sec. 5.

$(F, V, d, 0, I, M, \delta)^c$. The *initial global state* $\mathcal{S}_0 = \{\mathcal{C}_0\}$ consists of the *root component* $\mathcal{C}_0 = (F, V, 0, 0, \varepsilon, \perp, \infty)^o$ with F and V denoting the input formula and $V = \text{var}(F)$, 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_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:

$$\alpha: \mathcal{S} \uplus \{\mathcal{C}\} \rightsquigarrow_R \mathcal{S} \uplus \mathcal{C}', \beta: \mathcal{S} \uplus \{\mathcal{C}\} \rightsquigarrow_R \mathcal{S} \uplus \{\mathcal{C}', \mathcal{C}_1, \mathcal{C}_2\}, \gamma: \mathcal{S} \uplus \{\mathcal{C}, \mathcal{C}_1, \mathcal{C}_2\} \rightsquigarrow_R \mathcal{S} \uplus \{\mathcal{C}'\}.$$

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 . Finally, a γ rule removes 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 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 (`CompTrue`) or falsifies (`CompFalse`) its formula. 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 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, 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 adding multiple models of F to M in

Unit:	$\mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^o\} \rightsquigarrow_{\text{Unit}} \mathcal{S} \uplus \{(F, V, d, 0, I\ell, M, \delta[\ell \mapsto a])^o\}$ if $\perp \notin F _I$ and exists $C \in F$ with $\{\ell\} = C _I$ and $a \stackrel{\text{def}}{=} \delta(C \setminus \{\ell\})$
Decide:	$\mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^o\} \rightsquigarrow_{\text{Decide}} \mathcal{S} \uplus \{(F, V, d, 0, I\ell^d, M, \delta[\ell \mapsto a])^o\}$ if $F _I \neq \top$ and $\perp \notin F _I$ and $\text{units}(F _I) = \emptyset$ and $\text{var}(\ell) \in V$ 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 $\text{var}(G) \cap \text{var}(H) = \emptyset$
BackTrue:	$\mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^o\} \rightsquigarrow_{\text{BackTrue}} \mathcal{S} \uplus \{(F, V, d, 0, PK\ell, M \vee I, \delta[L \mapsto \infty][\ell \mapsto e])^o\}$ if $F _I = \top$ and $PQ \stackrel{\text{def}}{=} I$ and $D \stackrel{\text{def}}{=} \neg \text{decs}(I)$ and $e + 1 \stackrel{\text{def}}{=} \delta(D) = \delta(I)$ and $\ell \in D$ and $e = \delta(D \setminus \{\ell\}) = \delta(P)$ and $K \stackrel{\text{def}}{=} Q_{\leq e}$ and $L \stackrel{\text{def}}{=} Q_{> e}$
BackFalse:	$\mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^o\} \rightsquigarrow_{\text{BackFalse}} \mathcal{S} \uplus \{(F, V, d, 0, PK\ell, M, \delta[L \mapsto \infty][\ell \mapsto j])^o\}$ if exists $C \in F$ and exists D with $PQ \stackrel{\text{def}}{=} I$ and $C _I = \perp$ and $c \stackrel{\text{def}}{=} \delta(C) = \delta(D) > 0$ such that $\ell \in D$ and $\neg \ell \in \text{decs}(I)$ and $\ell _Q = \perp$ 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 $K \stackrel{\text{def}}{=} Q_{\leq b}$ and $L \stackrel{\text{def}}{=} Q_{> b}$
CompTrue:	$\mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^o\} \rightsquigarrow_{\text{CompTrue}} \mathcal{S} \uplus \{(F, V, d, 0, I, M \vee I, \delta)^c\}$ if $F _I = \top$ and $\text{decs}(I) = \emptyset$
CompFalse:	$\mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^o\} \rightsquigarrow_{\text{CompFalse}} \mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^c\}$ if exists $C \in F$ and $C _I = \perp$ and $\delta(C) = 0$
Decompose:	$\mathcal{S} \uplus \{(F, V, d, 0, I, M, \delta)^o\} \rightsquigarrow_{\text{Decompose}} \mathcal{S} \uplus \{(F, V, d, 2, I, M, \delta)^f, (G, U, d \cdot 1, 0, \varepsilon, \perp, \infty)^o, (H, W, d \cdot 2, 0, \varepsilon, \perp, \infty)^o\}$ if $F _I \neq \top$ and $\perp \notin F _I$ and $\text{units}(F _I) = \emptyset$ and $G \wedge H \stackrel{\text{def}}{=} F _I$ and $U \stackrel{\text{def}}{=} \text{var}(G)$ and $W \stackrel{\text{def}}{=} \text{var}(H)$ and $U \cap W = \emptyset$
ComposeBack:	$\mathcal{S} \uplus \{(F, V, d, 2, I, M, \delta)^f, (G, U, d \cdot 1, 0, J_G, N, \delta_G)^c, (H, W, d \cdot 2, 0, J_H, O, \delta_H)^c\} \rightsquigarrow_{\text{ComposeBack}} \mathcal{S} \uplus \{(F, V, l, 0, PK\ell, M \vee (I \wedge N \wedge O), \delta[L \mapsto \infty][\ell \mapsto e])^o\}$ if $PQ \stackrel{\text{def}}{=} I$ and $D \stackrel{\text{def}}{=} \neg \text{decs}(I)$ and $e + 1 \stackrel{\text{def}}{=} \delta(D) = \delta(I)$ and $\ell \in D$ and $e = \delta(D \setminus \{\ell\}) = \delta(P)$ and $K \stackrel{\text{def}}{=} Q_{\leq e}$ and $L \stackrel{\text{def}}{=} Q_{> e}$
ComposeEnd:	$\mathcal{S} \uplus \{(F, V, d, 2, I, M, \delta)^f, (G, U, d \cdot 1, 0, J_G, N, \delta_0)^c, (H, W, d \cdot 2, 0, J_H, O, \delta_0)^c\} \rightsquigarrow_{\text{ComposeEnd}} \mathcal{S} \uplus \{(F, V, d, 0, I, M \vee (I \wedge N \wedge O), \delta)^c\}$ if $\text{decs}(I) = \emptyset$

Fig. 2. ABSTRACT CNF2DDNNF transition rules.

one step, which 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 ($\delta(D) > 0$), 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 , either $N = \perp$ or $O = \perp$ and $I \wedge N \wedge O = \perp$. Otherwise, $I \wedge N \wedge O \neq \perp$. Both cases are captured with one single rule.

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 ($\text{decs}(I) = \emptyset$), 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 $I \wedge N \wedge O = \perp$ or $I \wedge N \wedge O \neq \perp$.

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 **ABSTRACT CNF2DDNNF** can not get stuck and terminates in a correct state. Showing termination concludes our proof.

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

- (1) $\forall k, \ell \in \text{decs}(I) . \tau(I, k) < \tau(I, \ell) \implies \delta(k) < \delta(\ell)$
- (2) $\delta(\text{decs}(I)) = \{1, \dots, \delta(I)\}$
- (3) $\forall n \in \mathbb{N} . F \wedge \neg M \wedge \text{decs}_{\leq n}(I) \models I_{\leq n}$, 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) $c > 0$ iff (A) $c = 2$, (B) \mathcal{C} is decomposed, and (C) \mathcal{S} contains two components $\mathcal{C}_G = (G, \text{var}(G), d \cdot 1, c_g, J_G, N, \delta_G)^s, \mathcal{C}_H = (H, \text{var}(H), d \cdot 2, c_H, J_H, O, \delta_H)^s$, such that $F|_I = G \wedge H$ and $\text{var}(G) \cap \text{var}(H) = \emptyset$
- (7) if $c = 2$ and \mathcal{S} contains two components $\mathcal{C}_G = (G, \text{var}(G), d \cdot 1, 0, J_G, N, \delta_G)^c$ and $\mathcal{C}_H = (H, \text{var}(H), d \cdot 2, 0, J_H, O, \delta_H)^c$, then $F|_I \equiv I \wedge N \wedge O$
- (8) If \mathcal{C} is closed, then $\text{decs}(I) = \emptyset$

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

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

Theorem 1 (Soundness of **ABSTRACT CNF2DDNNF Rules).** *The rules of **ABSTRACT CNF2DDNNF** 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 $\mathcal{C} = (F, V, d, c, I, M, \delta)^s$ and in the target state by $\mathcal{C}' = (F, V, d', c', I', M', \delta')^s$. Its subcomponents, if there are any, are written $\mathcal{C}_G = (G, \text{var}(G), d \cdot 1, c_G, J, N, \delta_G)^s, \mathcal{C}_H = (H, \text{var}(H), d \cdot 2, c_H, K, O, \delta_H)^s$.

Unit, Decide, BackTrue, and BackFalse: Apart from the additional elements V, d, c 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**, invariants (1) – (5) hold. Notice that in the proof of invariant (4), it suffices to replace “DSOP” by “d-DNNF”, since the relevant property here is determinism. Since $c' = 0$, invariants (6) and (7) do not apply. An open state is mapped to an open state, hence (8) holds.

CompTrue: Invariants (1) and (2) hold, since I remains unaffected. Since \mathcal{C}' is closed, invariants (3) and (4) are trivially met. We know that $M \vee F \wedge O(I) \equiv F$ holds and need to prove that $(M \vee I) \vee F \wedge O(I) \equiv F$ holds. Now $\text{decs}(I) = \emptyset$, hence $R(I) = \perp$, therefore $O(I) = I$. Since $I \models F$, we have that $F \wedge I \equiv I$, hence $(M \vee I) \vee F \wedge O(I) \equiv M \vee I \equiv F$ due to the premise, and invariant (5) holds. Since $c' = 0$ and $I' = I$, invariants (6) – (8) are trivially met.

CompFalse: Invariants (1) and (2) hold, since I remains unaltered. Since \mathcal{C}' is closed, invariants (3) and (4) are trivially satisfied. For showing that $M \vee F \wedge O(I) \equiv F$ holds, we observe that $R(I) = R_{\delta(\mathcal{C})}(I) = \perp$, due to $\text{decs}(I) = \emptyset$, and hence $O(I) = I$. Furthermore, $F \wedge I \equiv \perp$. Hence, $M \vee F \wedge O(I) \equiv M \equiv F$, due to the premise, and invariant (5) holds. Due to $c' = 0$ and $I' = I$ and hence $\text{decs}(I') = \emptyset$, invariants (6) – (8) are trivially satisfied.

Decompose: The parent component \mathcal{C} remains unaltered except for $c' = 2$ 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 $c_G = c_H = 0$. Therefore, invariants (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, invariants (7) and (8) hold.

ComposeBack: It suffices to show that the application of rule **ComposeBack** preserves the validity of the invariants for \mathcal{C}' , since \mathcal{C}_G and \mathcal{C}_H do not occur in the target state anymore. The most recent decision literal is flipped, similar to rule **BackTrue**. The same argument to the one given there applies, and invariants (1) and (2) are satisfied. We need to show that $F \wedge \neg(M \vee (I \wedge N \wedge O)) \wedge \text{decs}_{\leq n}(PK\ell) \models (PK\ell)_{\leq n}$ 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, $F \wedge \neg(M \vee (I \wedge N \wedge O)) \wedge \text{decs}_{\leq n}(PK\ell) \equiv (\neg I \wedge (F \wedge \neg M \wedge \text{decs}_{\leq n}(I))) \vee (F \wedge \neg M \wedge \neg(N \wedge O) \wedge \text{decs}_{\leq n}(I))$, since ℓ is not a decision literal in $PK\ell$ and $I_{\leq e} = PK$ and thus $I_{\leq n} = (PK)_{\leq n}$ by definition. By applying the induction hypothesis, we get $\neg I \wedge F \wedge \neg M \wedge \text{decs}_{\leq n}(PK\ell) \models (PK)_{\leq n}$, and hence $F \wedge \neg(M \vee (I \wedge N \wedge O)) \wedge \text{decs}_{\leq n}(PK\ell) \models (PK)_{\leq n}$. We still need to show that $F \wedge \neg(M \vee (I \wedge N \wedge O)) \wedge \text{decs}_{\leq e}(PK\ell) \models \ell$, 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 $F \wedge \neg \text{decs}_{\leq e}(PK) \wedge \neg \ell \models I$ and thus $F \wedge \neg \text{decs}_{\leq e}(PK) \wedge \neg I \models \ell$ by conditional contraposition, and invariant (3) holds.

For proving that invariant (4) holds, we consider two cases: (A) $I \wedge N \wedge O \neq \perp$, i.e., there exists an extension of I which satisfies F , and (B) $I \wedge N \wedge O = \perp$, 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 $O(I) = I \vee R_{\leq e+1}(I) = I \vee R_{\leq e}(I) \vee R_{=e+1}(I)$. The pending search space of $PK\ell$ is given by $O(PK\ell) = PK\ell \vee R_{\leq e}(PK\ell)$. But $PK = I_{\leq e}$ and $PK\ell = I_{\leq e}\ell = R_{=e+1}(I)$, since $\neg \ell \in \text{decs}(I)$ and $\delta(\neg \ell) = e + 1$. Furthermore, $R_{\leq e}(PK\ell) = R_{\leq e}(PK)$, since $\ell \notin \text{decs}(PK\ell)$ and $\delta(\ell) = e$, hence $R_{\leq e}(PK\ell) = R_{\leq e}(I)$. We have $O(PK\ell) = R_{=e+1}(I) \vee R_{\leq e}(I)$, 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 $M \vee O(PK\ell) = M \vee R_{\leq e+1}(I)$. Recalling that $R_{\leq e+1}(I)$ 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, invariant (4) holds after the application of rule `ComposeBack`.

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

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

(B) We must show that $M \vee (F \wedge O(PK\ell)) \equiv F$. Similarly to (A) we have $M \vee (F \wedge O(I)) \equiv M \vee (F \wedge O(PK\ell)) \vee (F \wedge I) \equiv M \vee (F \wedge O(PK\ell)) \equiv F$, due to $F \wedge I \equiv F$. Therefore, invariant (5) holds after applying rule `ComposeBack`. We have $c' = 0$, and \mathcal{C}' is open, hence invariants (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 $\text{decs}(I) = \emptyset$ and since \mathcal{C}' is closed, invariants (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) $I \wedge N \wedge O \neq \perp$ and (B) $I \wedge N \wedge O = \perp$.

(A) From $\text{decs}(I) = \emptyset$, 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. Hence, invariant (5) holds after executing rule `ComposeEnd`. Invariants (6) – (8) trivially hold, due to $c' = 0$ and $I' = I$ and hence $\text{decs}(I') = \emptyset$.

Corollary 1 (Soundness of ABSTRACT CNF2DDNNF Run). *ABSTRACT CNF2DDNNF 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 invariant (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 ABSTRACT CNF2DDNNF $M \equiv F$ 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). **ABSTRACT CNF2DDNNF** *always makes progress.*

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.

First the case of a component without subcomponents is considered. Let a global state be $\mathcal{S} \uplus \{\mathcal{C}\}$, where $\mathcal{C} = (F, V, d, 0, I, M, \delta)^o$, and assume $F|_I = \top$. If $\text{decs}(I) = \emptyset$, rule **CompTrue** is applicable. Otherwise, we choose $D = \neg \text{decs}(I)$. We have $\delta(D) = \delta(I) = e + 1$ and due to invariant (2), the clause D contains exactly one literal ℓ such that $\delta(\ell) = e + 1$ and therefore $\delta(D \setminus \{\ell\}) = e$. Now we choose P and Q such that $I = PQ$ and $\delta(P) = e$ and in particular $\ell|_P = \perp$. After backtracking to decision level e , we have $I_{\leq e} = PK$, where $K = Q_{\leq e}$ and $D|_{PK} = \{\ell\}$, and all preconditions of **BackTrue** are met.

If instead $F|_I = \perp$, there is a clause $C \in F$ such that $C|_I = \perp$. If $\delta(C) = c = 0$, rule **CompFalse** is applicable. Otherwise, by invariant (3) we have $F \wedge \neg M \wedge \text{decs}_{\leq c}(I) \equiv (F \wedge \neg M \wedge \text{decs}_{\leq c}(I) \wedge I_{\leq c} \models I_{\leq c})$. Since $I_{\leq c}(F) \equiv \perp$, also $F \wedge \neg M \wedge \text{decs}_{\leq c}(I) \wedge I_{\leq c} \equiv F \wedge \neg M \wedge \text{decs}_{\leq c}(I) \equiv \perp$. We choose $\neg D = \text{decs}(I)$ obtaining $F \wedge \neg M \wedge \neg D \wedge I_{\leq c} \equiv \perp$, thus $F \wedge \neg M \models D$. Similarly to the proof of progress given in Proposition 2 in our formalization of chronological CDCL [33], but for $b = c - 1$, it can be shown that all preconditions of rule **BackFalse** hold. Notice that D is not added explicitly to F . Instead, the clause D is used exclusively to determine ℓ .

Finally, if $F|_I \notin \{\top, \perp\}$, there are still unassigned variables. If there exists a clause $C \in F$ such that $C|_I = \{\ell\}$, the preconditions of rule **Unit** are satisfied. Otherwise, if $F|_I$ can be decomposed into formulae G and H where $\text{var}(G) \cap \text{var}(H) = \emptyset$, the preconditions of rule **Decompose** are satisfied. Otherwise, there is a literal ℓ with $\text{var}(\ell) \in V$ and $\delta(\ell) = \infty$, and the preconditions of **Decide** hold.

Now suppose there exists a decomposed component, i.e., let a global state be given by $\mathcal{S} \uplus \{\mathcal{C}\}$ where $\mathcal{C} = (F, V, d, 2, I, M, \delta)^f$. Due to invariant (6), \mathcal{S} contains two (sub)components $\mathcal{C}_G = (G, \text{var}(G), d \cdot 1, c_g, J_G, N, \delta_G)^s$ and $\mathcal{C}_H = (H, \text{var}(H), d \cdot 2, c_h, J_H, O, \delta_H)^s$ such that $F|_I = G \wedge H$ and $\text{var}(G) \cap \text{var}(H) = \emptyset$. Assume $s = c$ for both \mathcal{C}_G and \mathcal{C}_H , i.e., both \mathcal{C}_G and \mathcal{C}_H are closed, and $c_G = c_H = 0$. If $\text{decs}(I) = \emptyset$, rule **ComposeEnd** is applicable. Otherwise, with the same argument as for rule **BackTrue**, we can show that all preconditions of rule **ComposeBack** are satisfied. Now suppose $s \in \{f, o\}$ for at least one of \mathcal{C}_G and \mathcal{C}_H , i.e., at least one of \mathcal{C}_G and \mathcal{C}_H is not closed. Then the non-closed component(s) can be further processed, and as soon as both \mathcal{C}_G and \mathcal{C}_H are closed, rule **ComposeEnd** or **ComposeBack** can be applied. This proves that **ABSTRACT CNF2DDNNF** always makes progress.

Unit: $\mathcal{S}' \uplus \{(d, [l_1, \dots, l_k, 2, 2, \dots, 2], o)\}$ \succ_{ACD} $\mathcal{S}' \uplus \{(d, [l_1, \dots, l_k, 0, 2, \dots, 2], o)\}$	Decide: $\mathcal{S}' \uplus \{(d, [l_1, \dots, l_k, 2, 2, \dots, 2], o)\}$ \succ_{ACD} $\mathcal{S}' \uplus \{(d, [l_1, \dots, l_k, 1, 2, \dots, 2], o)\}$
BackTrue: $\mathcal{S}' \uplus \{(d, [l_1, \dots, l_k, 1, l_{k+2}, \dots, l_{ V }], o)\}$ \succ_{ACD} $\mathcal{S}' \uplus \{(d, [l_1, \dots, l_k, 0, l'_{k+2}, \dots, l'_{ V }], o)\}$	BackFalse: $\mathcal{S}' \uplus \{(d, [l_1, \dots, l_k, 1, l_{k+2}, \dots, l_{ V }], o)\}$ \succ_{ACD} $\mathcal{S}' \uplus \{(d, [l_1, \dots, l_k, 0, l'_{k+2}, \dots, l'_{ V }], o)\}$
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: $\mathcal{S}' \uplus \{(d, t, o)\} \succ_{\text{ACD}} \mathcal{S}' \uplus \{(d, t, f), (d \cdot 1, [2, \dots, 2], o), (d \cdot 2, [2, \dots, 2], o)\}$	
ComposeBack: $\mathcal{S}' \uplus \{(d, [l_1, \dots, l_k, 1, l_{k+2}, \dots, l_{ V }], f), (d \cdot 1, t_1, c), (d \cdot 2, t_2, c)\}$ \succ_{ACD} $\mathcal{S}' \uplus \{(d, [l_1, \dots, l_k, 0, l'_{k+2}, \dots, l'_{ V }], o)\}$	
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. 3. Rule applications lead to smaller global states.

Theorem 4 (Termination). ABSTRACT CNF2DDNNF *always terminates*.

Proof. We need to show that no infinite rule applications can happen. To this end, we define a strict, well-founded ordering \succ_{ACD} on the global states and show that $\mathcal{S} \rightsquigarrow_{\text{R}} \mathcal{T}$ implies $\mathcal{S} \succ_{\text{ACD}} \mathcal{T}$ for all $\mathcal{S}, \mathcal{T} \in S$ and rules in ABSTRACT CNF2DDNNF. Global states are sets of components, and \succ_{ACD} is the multiset extension of a component ordering $\succ_{\text{c}} = (\succ_{\text{cl}}, \succ_{\text{tr}}, \succ_{\text{cs}})$, where \succ_{cl} , \succ_{tr} , and \succ_{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 as lists over $\{0, 1, 2\}$. A trail $I = l_1 \dots l_k$ defined over V , where $k \leq |V|$, is represented as $[l_1, \dots, l_k, 2, \dots, 2]$, 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 $>_{\text{lex}}$ on trails, and we define \succ_{tr} as the restriction of $>_{\text{lex}}$ to $\{[l_1, \dots, l_{|V|}] \mid l_i \in \{0, 1, 2\} \text{ for } 1 \leq i \leq |V|\}$, i.e., we have $t_1 \succ_{\text{tr}} t_2$ if $t_1 >_{\text{lex}} t_2$. The ordering \succ_{tr} is well-founded, its minimal element is $[0, \dots, 0]$.

The component state takes values in $\{o, f, c\}$, and we define \succ_{cs} as $>_{\text{lex}}$, i.e., $s_1 \succ_{\text{cs}} s_2$ if $s_1 >_{\text{lex}} s_2$. The minimal element of \succ_{cs} is c , hence \succ_{cs} is well-founded.

Given two component levels d_1 and d_2 , we define $d_1 \succ_{\text{cl}} d_2$ if $\text{length}(d_1) < \text{length}(d_2)$. This may seem somewhat counterintuitive but is needed to avoid that the execution of rule Decompose results in a higher state due to the newly created

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

Fig. 4. Generalized transition rules.

(sub)components, since both their component state and their trail are of higher order than those of their parent component. To see that \succ_{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_{cs} .

Now we have all ingredients for the component ordering $\succ_c = (\succ_{\text{cl}}, \succ_{\text{tr}}, \succ_{\text{cs}})$. Let components \mathcal{C}_1 and \mathcal{C}_2 be encoded as (d_1, t_1, s_1) and (d_2, t_2, s_2) , respectively. We have $\mathcal{C}_1 \succ_c \mathcal{C}_2$ if $\mathcal{C}_1 \neq \mathcal{C}_2$ and $d_1 \succ_{\text{cl}} d_2$ or $d_1 = d_2$ and either $t_1 \succ_{\text{tr}} t_2$ or $t_1 = t_2$ and $s_1 \succ_{\text{cs}} s_2$. Clearly \succ_c is well-founded, since \succ_{tr} , \succ_{cs} , and \succ_{cl} are well-founded. Finally, for two global states \mathcal{S} and \mathcal{T} , we have $\mathcal{S} \succ_{\text{ACD}} \mathcal{T}$ 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_c , \mathcal{S} contains a component \mathcal{C}' that is larger in \mathcal{S} than in \mathcal{T} . Since \succ_c is well-founded, also \succ_{ACD} is well-founded. Figure 3 shows that each rule application leads to a smaller global state, which concludes our proof.

6 Generalization

For the sake of simplicity, we have presented the case in which a formula is partitioned into two subformulae. However, lifting our calculus to an arbitrary number of subcomponents is straightforward. The generalized rules capturing component-based reasoning are listed in Fig. 4. In our generalized framework, we have $F|_I = \bigwedge_{i=1}^n G_i$, and accordingly the G_i are required to be defined over pairwise disjoint sets of variables (rule DecomposeG). As in ABSTRACT CNF2DDNNF, rules ComposeBackG and ComposeEndG are applicable as soon as all subcomponent states are closed.

7 Discussion

We have presented ABSTRACT CNF2DDNNF, 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]. 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 result in shrinking a larger part of the search space processed by a single computation. Our framework lays the theoretical foundation for practical All-SAT and #SAT solving. Any implementation which can be modeled by ABSTRACT CNF2DDNNF exhibits its properties, in particular its correctness, which has been established in a rigorous 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 ABSTRACT CNF2DDNNF, both D4 and Dsharp adopt a dynamic decomposition strategy, while C2D constructs a decomposition tree which it then uses 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, and from this point of view, partial models are superior to total models. The dual reasoning approach 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 method lies in an efficient implementation of the oracle calls required by the entailment checks. It also would be interesting to investigate the impact of dynamic component analysis on a recent implementation [46] of the former framework addressing model enumeration by means of chronological CDCL [34]. Cache structures, being an inherent part of modern knowledge compilers and #SAT solvers [11,16,19,20,31,30,37,41,42,47,49] 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 real-world 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., Fahrenholtz, 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: SAT. *Lecture Notes in Computer Science*, vol. 9340, pp. 199–214. Springer (2015)
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: SAT. *Lecture Notes in Computer Science*, vol. 9340, pp. 46–61. Springer (2015)
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: SAT. *Lecture Notes in Computer Science*, vol. 4121, pp. 61–74. Springer (2006)
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 form. *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: *SAT. Lecture Notes in Computer Science*, vol. 11628, pp. 250–266. Springer (2019)
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: *SAT. Lecture Notes in Computer Science*, vol. 12178, pp. 62–71. Springer (2020)
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.J., McIlraith, S.A., Beck, J.C., Hsu, E.I.: Dsharp: Fast d-DNNF compilation with sharpSAT. In: *Canadian Conference on AI. Lecture Notes in Computer Science*, vol. 7310, pp. 356–361. Springer (2012)
38. Nadel, A., Ryvchin, V.: Chronological backtracking. In: *SAT. Lecture Notes in Computer Science*, vol. 10929, pp. 111–121. Springer (2018)
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. Silva, J.P.M., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: *ICCAD*. pp. 220–227. *IEEE Computer Society / ACM* (1996)
45. Silva, J.P.M., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers* **48**(5), 506–521 (1999)

46. Spallitta, G., Sebastiani, R., Biere, A.: Enumerating disjoint partial models without blocking clauses, manuscript submitted for publication
47. Thurley, M.: sharpSAT – counting models with advanced component caching and implicit BCP. In: SAT. Lecture Notes in Computer Science, vol. 4121, pp. 424–429. Springer (2006)
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: SAT. Lecture Notes in Computer Science, vol. 2919, pp. 287–298. Springer (2003)