IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 27, 2019, accepted July 1, 2019, date of publication July 5, 2019, date of current version July 25, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2927041

Augmenting Negation Normal Form With Irrelevant Variables

DANGDANG NIU[1,2,3,4], LEI LIU[2], AND SHUAI LÜ[2,4]

[1]College of Information Engineering, Northwest A&F University, Yangling 712100, China
[2]College of Computer Science and Technology, Jilin University, Changchun 130012, China
[3]Shaanxi Key Laboratory of Agricultural Information Perception and Intelligent Service, Northwest A&F University, Yangling 712100, China
[4]Key Laboratory of Symbolic Computation and Knowledge Engineering, Ministry of Education, Jilin University, Changchun 130012, China

Corresponding author: Shuai Lü (lus@jlu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61763003, in part by the Natural Science Research Foundation of Jilin Province of China under Grant 20180101053JC, and in part by the Fundamental Research Funds for the Central Universities, Jilin University.

ABSTRACT Irrelevant variables are always omitted in knowledge compilation languages since their assignments do not change the satisfiability of sentences. In order to identify new knowledge compilation languages and reduce the scale of compiling result of d-DNNF, we augment NNF with irrelevant variables in this paper. The $\mathrm{NNF_{PNI}}$, $\mathrm{NNF_{PI}}$, and $\mathrm{NNF_{NI}}$ are proposed based on different combinations of positive literals, negative literals, and irrelevant variables. Each sentence in NNF, $\mathrm{NNF_{PI}}$, $\mathrm{NNF_{NI}}$, or $\mathrm{NNF_{PNI}}$ can be translated to an equivalent sentence in another language in polynomial time. We also define $\mathrm{d\text{-}DNNF_{NI}}$, $\mathrm{d\text{-}DNNF_{PI}}$, and $\mathrm{d\text{-}DNNF_{PNI}}$ based on decomposability and determinism in NNF, which are subclasses of $\mathrm{NNF_{PI}}$, $\mathrm{NNF_{NI}}$, and $\mathrm{NNF_{PNI}}$, respectively. A number of querying and transforming methods for $\mathrm{d\text{-}DNNF_{PNI}}$, $\mathrm{d\text{-}DNNF_{PI}}$, and $\mathrm{d\text{-}DNNF_{NI}}$ are designed to solve relevant reasoning problems in knowledge compilation map. Overall, $\mathrm{d\text{-}DNNF_{PI}}$ and $\mathrm{d\text{-}DNNF_{NI}}$ do not reduce the tractability of d-DNNF, so we propose a compressing method for d-DNNF based on $\mathrm{d\text{-}DNNF_{PI}}$ and $\mathrm{d\text{-}DNNF_{NI}}$. The experimentally, the compiling results of the $\mathrm{d\text{-}DNNF_{PI}}$ and $\mathrm{d\text{-}DNNF_{NI}}$ are better with respect to d-DNNF for most instances, and our compressing method is significantly effective for all instances.

INDEX TERMS Knowledge compilation, negation normal form, d-DNNF, irrelevant variables.

I. INTRODUCTION

Knowledge compilation has been emerging as a popular direction of research for improving the efficiency of computational tasks. According to this direction, the reasoning process is split into two phases: an off-line compilation phase and an on-line reasoning phase. In the off-line phase, the propositional theory is compiled into some target compilation language. In the on-line phase, the compiled target is used to answer a large number of queries in polytime. This paper is primarily concerned with reducing the scale of compiling result of target compilation language, which can further improve the efficiency of on-line reasoning.

Over the years, many target compilation languages have been proposed and researched for different inference tasks, including ROBDDs [1], prime implicates [2], prime implicants [3], DNNF [4], FNNF [5], AOMDD [6], $\mathrm{DNNF_T}$ [7], SDDs [8], ZSDDs[9], OBDD-L [10], and OBDD[$\wedge$] [11]. In order to evaluate different target compilation languages, Darwiche and Marquis have provided the knowledge compilation map by analyzing them according to their succinctness and the class of queries/transformations that they support in polytime [12].

New target compilation languages are usually identified by augmented existing languages with appropriate knowledge representation properties. For example, SDDs is the results from imposing structured decomposability and strong determinism on decision diagram [8], and ZSDDs is identified from augmenting SDDs with the Zero-suppressed property [9]. New target compilation languages can improve existing knowledge compilation methods from different aspects. Likewise, ZSDDs can be more succinct than SDDs when representing sparse Boolean functions [9], and ROBDD[$\wedge$] can admit more transformations than ROBDD in polytime [11].

Given a formula F, a partial assignment P assigns the variables mentioned by it in F to *true* or *false*. If $F|P$ is *true* or *false*, the variables not mentioned by P are irrelevant. Usually, irrelevant variables are omitted in knowledge

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Afzal.

compilation languages, since thy can be computed based on known positive literals and negative literals. A natural idea is that we can also omit positive literals (resp. negative literals) and compute them based on known negative literals (resp. positive literals) and irrelevant variables in knowledge compilation languages. So, irrelevant variable can be viewed as a new appropriate knowledge presentation property, and it can be used to identify new target compilation languages.

Negation Normal Form, NNF, is a nested class based on representing propositional sentences using directed acyclic graphs. A number of target compilation languages that have been presented in the AI, formal verification, and computer science literature and show that they are special cases of NNF [12]. In a sentence in NNF, each leaf nodes is labeled with *true*, *false*, positive literal or negative literal. Obviously, NNF also omits irrelevant variables.

Deterministic, Decomposable Negation Normal Form, d-DNNF [13], is a representative subclass of NNF and a tractable logical form, which permits some generally intractable logical queries to be computed in polynomial time in the form size [12], [13]. d-DNNF has been successfully employed in many applications, including probabilistic reasoning [14]–[16], diagnosis [17], and Max-SAT [18]. c2d is a publicly available compiler that converts CNF to d-DNNF [19]. Afterwards, an efficient d-DNNF compiler DSHARP is proposed based on sharpSAT [20]. Recently, some new technologies for compiling a CNF to decision-DNNF (a strict subset of d-DNNF) are proposed [21].

Knowledge compilation pushes much of the computational overhead into off-line compilation, which is amortized over all on-line queries. Therefore, the efficiency of on-line reasoning is a crucial factor for knowledge compilation, which is directly decided by the scale of compiling result of target compilation language. d-DNNF is one of the most important target compilation languages and has a number of important applications in reasoning field, which motivates us to further improve the performance of knowledge compilation methods based on d-DNNF. We focus on improving the efficiency of on-line reasoning by reducing the scale of compiling result for d-DNNF in this paper.

Irrelevant variables can be used to equivalently replace positive literals or negative literals in target compilation languages. For most real-world problems, the number of irrelevant variables is far less with respect to positive literals and negative literals in their models. However, irrelevant variables are omitted in NNF. So, we intend to augment NNF with irrelevant variables and to identify new knowledge compilation languages in this paper. In this way, we can further reduce the scale of compiling result of d-DNNF by replacing their positive literals or negative literals with irrelevant variables.

In this paper, we augment NNF with irrelevant variables, and NNF_{PNI}, NNF_{PI} and NNF_{NI} are proposed.

- In an $\mathbf{NNF_{PNI}}$ sentence, each leaf node of each sentence is labeled by *true*, *false*, some positive literal, some negative literal, or some irrelevant variable.
- In an $\mathbf{NNF_{PI}}$ sentence, each leaf node of each sentence is labeled by *true*, *false*, some positive literal, or some irrelevant variables.
- In an $\mathbf{NNF_{NI}}$ sentence, each leaf node of each sentence is labeled by *true*, *false*, some negative literal, or some irrelevant variables.

Each sentence in NNF, NNF_{PI} or NNF_{NI} can be translated to an equivalent sentence in NNF_{PNI} in polynomial time, vice-versa. We also design a number of querying and transforming methods for d-DNNF_{PNI}, d-DNNF_{PI} and d-DNNF_{NI} to solve reasoning problems in knowledge compilation map [12]. Additionally, we propose a compressing method for d-DNNF based on d-DNNF_{PI} and d-DNNF_{NI}. In order to evaluate the above new languages, we select a number of instances, which are encoded from real world problems and widely used in domain of knowledge compilation [8]–[11], [21]. Experimental results show that our compressing method can sharply reduce the scale of sentences in d-DNNF for a number of benchmarks.

II. BASIC DEFINITIONS

Definition 1 [13]: Let PS be a denumerable set of propositional variables. A sentence in NNF_{PS} is a rooted, directed acyclic graph (DAG) where each leaf node is labeled with $true, false, X$ or $\neg X, X \in PS$; and each internal node is labeled with $\wedge$ or $\vee$ and can have arbitrarily many children. The size of a sentence Σ in NNF_{PS}, denoted $|\Sigma|$, is the number of its DAG edges. Its height is the maximum number of edges from the root to some leaf in the DAG.

A number of properties can be stated on NNF graphs [13]:

- **Decomposability**. An NNF satisfies this property if for each conjunction C in the NNF, the conjuncts of C do not share variables. That is, if $C_1,\ldots,C_n$ are the children of and-node C, then $Vars(C_i) \cap Vars(C_j) = \emptyset$ for $i \neq j$.
- **Determinism**. An NNF satisfies this property if for each disjunction C in the NNF, each two disjuncts of C are logically contradictory. That is, if $C_1,\ldots,C_n$ are the children of or-node C, then $C_i \wedge C_j| = false$ for $i \neq j$.
- **Smoothness**. An NNF satisfies this property if for each disjunction C in the NNF, each disjunct of C mentions the same variables. That is, if $C_1,\ldots,C_n$ are the children of or-node C, then $Vars(C_i) = Vars(C_j)$ for $i \neq j$.

Definition 2 [13]: d-DNNF is the subset of NNF satisfying decomposability and determinism; and sd-DNNF is the subset satisfying decomposability, determinism and smoothness.

III. DEFINITIONS OF NEW COMPILATION LANGUAGES

Definition 3 (Irrelevant Variables): Given a sentence Σ in NNF, for each disjunction C in Σ, if there exists a disjunct D of C with $Vars(D) \neq Vars(C)$, the variables in $Vars(C)$ - $Vars(D)$ are irrelevant variables with respect to D (denoted as irrelevant variables for convenience).

Usually, an assignment consists of positive literals, negative literals and irrelevant variables, where irrelevant variables are default. We can also make the positive literals or negative literals to be default.

Example 1: Given a CNF formula $F = \{\neg A \vee B \vee C, \neg A \vee C, B \vee \neg C, D\}$ and $s = \{\neg A, \neg C, D\}$ the satisfiable assignment of F. According to s, we know that B is an irrelevant variable with respect to s. B_I represents that B is an irrelevant variable. If we make the positive literals to be default, we can get an assignment $s_1 = \{\neg A, B_I, \neg C\}$, in which D is default. If we make the negative literals to be default, we can get an assignment $s_2 = \{B_I, D\}$, in which $\neg A$ and $\neg C$ are default.

Definition 4 (NNF_{PNI}): Let PS be a denumerable set of propositional variables. A sentence in NNF_{PNI} is a rooted, directed acyclic graph (DAG) where each leaf node is labeled with *true*, *false*, X, $\neg X$ or X_I, $X \in PS$, X_I means that X is irrelevant; and each internal node is labeled with $\wedge$ or $\vee$ and can have arbitrarily many children.

Definition 5 (NNF_{PI}): Let PS be a denumerable set of propositional variables. A sentence in NNF_{PI} is a rooted, directed acyclic graph (DAG) where each leaf node is labeled with *true*, *false*, X or X_I, $X \in PS$, X_I means that X is irrelevant; and each internal node is labeled with $\wedge$ or $\vee$ and can have arbitrarily many children.

Definition 6 (NNF_{NI}): Let PS be a denumerable set of propositional variables. A sentence in NNF_{NI} is a rooted, directed acyclic graph (DAG) where each leaf node is labeled with *true*, *false*, $\neg X$ or X_I, $X \in PS$, X_I means that X is irrelevant; and each internal node is labeled with $\wedge$ or $\vee$ and can have arbitrarily many children.

Just like NNF, the size of a sentence Σ in NNF_{PNI}, NNF_{PI} or NNF_{NI}, denoted $|\Sigma|$, is the number of its DAG edges. Its height is the maximum number of edges from the root to some leaf in the DAG.

For NNF_{PNI}, NNF_{PI} and NNF_{NI}, all irrelevant variables are identified. So, for each disjunction C in NNF_{PNI}, each disjunct of C in NNF_{PNI} mentions the same variables. Specially, NNF_{PNI} satisfies smoothness.

Theorem 1: Every sentence in NNF, NNF_{PI} or NNF_{NI} can be translated to an equivalent sentence in NNF_{PNI} in polynomial time. Every sentence in NNF_{PNI} can be translated to an equivalent sentence in NNF, NNF_{PI}, or NNF_{NI} in polynomial time.

Proof: We can easily make a sentence in NNF_{PI} (resp. NNF_{NI} and NNF) smooth using the following operation: For each disjunction $\wedge_i \alpha_i$, replace the disjunct α_i with $\alpha_i \wedge \wedge_{A \in N} \neg A$(resp. A and A_I), where N are the atoms appearing in $\wedge_i \alpha_i$ but not in α_i. Above smoothing processing can be done in polynomial time. Every sentence in NNF_{PNI} can be translated to an equivalent sentence in NNF, NNF_{PI} or NNF_{NI} by replacing the negative literals, positive literals or irrelevant variables with the *true* node. Above processing can be done in linear time. So, Theorem 1 is established.

After replacing the negative literals or positive literals with the *true* node, there may exist some reducible edges under some *and* nodes. We use following reducing strategies to remove those reducible edges.

1) Removing all edges from *and* nodes to *true* node;

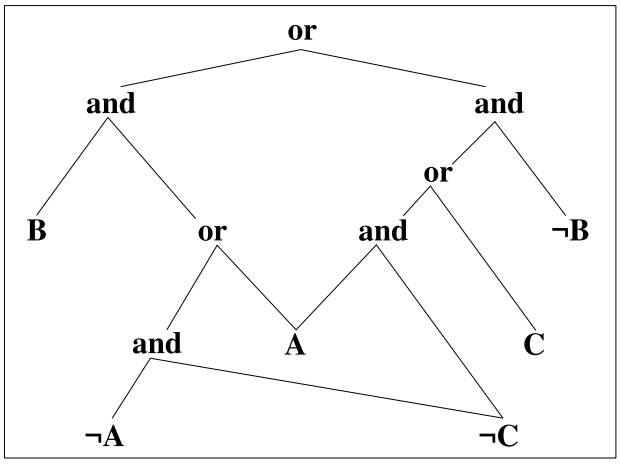

FIGURE 1. A sentence in d-DNNF. The size of this sentence is 14.

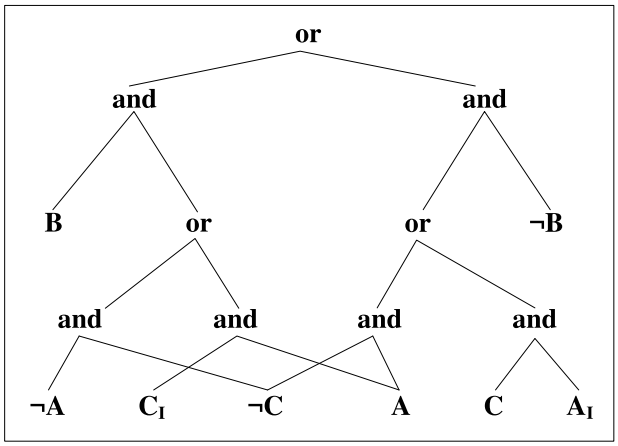

FIGURE 2. A sentence in d-$DNNF_{PNI}$, which is equivalent to the sentence in Fig. 1. C_I and A_I are two leaf nodes labeled with irrelevant variables. The size of this sentence is 18.

2) If the number of child nodes of an *and* node V is 0, labeling V with *true*;

3) If the number of child nodes of an *and* node V is 1, replacing V with its child node.

The definitions of decomposability and determinism in NNF are also applicable for NNF_{PNI}, NNF_{PI} and NNF_{NI}. It should be noted that the determinism in NNF_{PI} or NNF_{NI} should be considered with the omitted negative literals or positive literals.

Definition 7: d-$DNNF_{PNI}$ (reps. d-$DNNF_{PI}$ and d-$DNNF_{NI}$) is the subset of NNF_{PNI} (reps. NNF_{PI} and NNF_{NI}) satisfying decomposability and determinism.

d-$DNNF_{PNI}$ also satisfies smoothness.

Based on Theorem 1, we can reasonably infer that d-DNNF, d-$DNNF_{PI}$, d-$DNNF_{NI}$ and d-$DNNF_{PNI}$ have the same tractability with respect to some generally intractable logical queries and transformations in the form size.

Figures 2-4 are three sentences in d-$DNNF_{PNI}$, d-$DNNF_{PI}$ and d-$DNNF_{NI}$ respectively, which are equivalent to the sentence in Figure 1.

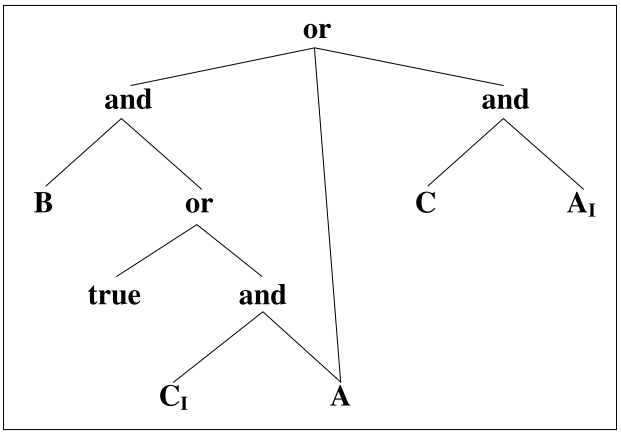

FIGURE 3. A sentence in d-DNNF$_{PI}$, which is equivalent to the sentence in Fig. 1. The size of this sentence is 11.

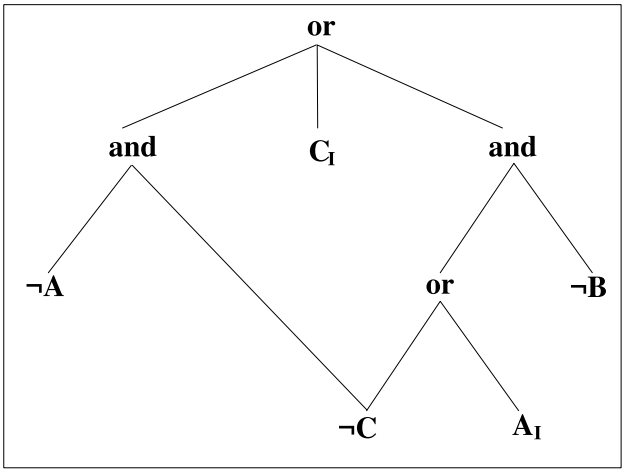

FIGURE 4. A sentence in d-DNNF$_{NI}$, which is equivalent to the sentence in Fig. 1. The size of this sentence is 9.

IV. REASONING ONd-DNNF$_{PNI}$, d-DNNF$_{PI}$ AND d-DNNF$_{NI}$

d-DNNF permits many logical queries to be computed in polynomial time in the form size. In knowledge compilation map, these queries include consistency check, validity check, clausal entailment check, implicant check, model counting and model enumeration [12]. And d-DNNF also supports conditioning transformation in polynomial time in the form size [12]. Given a sentence Σ, the definitions of queries and transformation mentioned in this paper are shown as follows.

- Consistency (validity) check: Mapping Σ to *true* if Σ is consistent (valid), to *false* otherwise.
- Clausal entailment check: For any clause C, mapping Σ and C to *true* if $\Sigma| = C$ holds, and to *false* otherwise.
- Implicant check: For any term T, mapping Σ and T to *true* if $T| = \Sigma$ holds, and to *false* otherwise.
- Model counting: Mapping Σ to a nonnegative integer that represents the number of models of Σ.
- Model enumeration: Outputting all models of Σ.
- Conditioning: For any consistent term T, mapping Σ and T to $\Sigma|T$.

Sentences in d-DNNF$_{PNI}$, d-DNNF$_{PI}$ and d-DNNF$_{NI}$ contain irrelevant variables. Sentences in d-DNNF$_{PI}$ and d-DNNF$_{NI}$ do not mention negative literals or positive literals. The above properties caused that original reasoning methods on d-DNNF are not applicable for d-DNNF$_{PNI}$, d-DNNF$_{PI}$ and d-DNNF$_{NI}$. So in this section, we intend to provide the methods of online reasoning for d-DNNF$_{PNI}$, d-DNNF$_{PI}$ and d-DNNF$_{NI}$.

Reasoning on d-DNNF$_{PNI}$, d-DNNF$_{PI}$ and d-DNNF$_{NI}$ can be implemented by translating them to d-DNNF, then calling corresponding reasoning methods for d-DNNF. But the translating processes needs a lot of time, though they can be done in polynomial time. So we design some new query and transformation methods for d-DNNF$_{PNI}$, d-DNNF$_{PI}$ and d-DNNF$_{NI}$ in this section. These new methods can take advantage of irrelevant variables, or be irrelevant to positive literals and negative literals.

Theorem 2: Given a consistent term T, an irrelevant variable X_I is consistent with instantiation T iff $X \in Vars(T)$.

Given a consistent term T, conditioning replaces every leaf node in Σ with *true* (*false*) if it is consistent (inconsistent) with instantiation T. The consistency of irrelevant variables with respect to any instantiation can be decided by Theorem 2. So conditioning on d-DNNF$_{PNI}$ can be done in linear time. But for d-DNNF$_{PI}$ (resp. d-DNNF$_{NI}$), the negative literals (resp. positive literals) are default, so the conditioning on d-DNNF$_{PI}$ (resp. d-DNNF$_{NI}$) sentences should be done on the corresponding d-DNNF$_{PNI}$ sentences. According to Theorem 1, every sentence d-DNNF$_{PI}$ or d-DNNF$_{NI}$ can be translated to an equivalent sentence in d-DNNF$_{PNI}$ in polynomial time. Therefore, conditioning on d-DNNF$_{PI}$ or d-DNNF$_{NI}$ based on d-DNNF$_{PNI}$ can be done in polynomial time.

Unlike d-DNNF, irrelevant variables may appear in the leaf nodes of a sentence in d-DNNF$_{PNI}$, d-DNNF$_{PI}$ or d-DNNF$_{NI}$. If D_i is labeled with an irrelevant variable, the sentence rooted at D_i is satisfiable. So we give the reasoning method of consistency for d-DNNF$_{PNI}$, d-DNNF$_{PI}$ and d-DNNF$_{NI}$ in Definition 8.

Definition 8: For a d-DNNF$_{PNI}$ (resp. d-DNNF$_{PI}$ or d-DNNF$_{NI}$) sentence Σ rooted at Δ, Consistency(Δ) is defined as follows:

- Consistency(Δ) = *false*, if Δ is labeled with *false*;
- Consistency(Δ) = *true*, if Δ is labeled with a literal, an irrelevant variable or *true*;
- Consistency(Δ) = $\vee_i$ Consistency(Δ), if Δ is labeled with *or*, where Δ_i are the children of Δ;
- Consistency(Δ) = $\wedge_i$ Consistency(Δ), if Δ is labeled with *and*, where Δ_i are the children of Δ.

Model counting on a d-DNNF sentence should be computed based on its equivalent sd-DNNF sentence. But for d-DNNF$_{PNI}$, d-DNNF$_{PI}$ and d-DNNF$_{NI}$, model counting can be directly computed.

Definition 9: For a d-DNNF$_{PNI}$ (resp. d-DNNF$_{PI}$ or d-DNNF$_{NI}$) sentence Σ rooted at Δ, ICount(Δ) is defined as follows:

• ICount (Δ) = 0, if Δ is labeled with *false*;
• ICount (Δ) = 1, if Δ is labeled with a literal or *true*;
• ICount (Δ) = 2, if Δ is labeled with an irrelevant variable;
• ICount (Δ) = $\sum_i$ ICount(Δ), if Δ is labeled with *or*, where Δ_i are the children of Δ;
• ICount (Δ) = $\prod_i$ ICount(Δ), if Δ is labeled with *and*, where Δ_i are the children of Δ.

Validity check of a d-DNNF sentence is realized based on counting models. For d-DNNF$_{\text{PNI}}$, d-DNNF$_{\text{PI}}$ and d-DNNF$_{\text{NI}}$, validity check is also completed based on counting models. The direct computing method is that: Given a sentence $\Sigma = (V, E, M)$ in d-DNNF$_{\text{PNI}}$, d-DNNF$_{\text{PI}}$ or d-DNNF$_{\text{NI}}$, in which V, E and M collect the vertices, edges and variables in Σ respectively, Σ is valid if the number of models of Σ is equal to $2^{|M|}$, and Σ is not valid otherwise. In the process of counting models of Σ, if the variable sets of all *or* node in Σ are known, we can also make the following decision for each *or* node O_i: If ICount(O_i) $\neq 2^{|Vars(Oi)|}$, Σ is not valid. Both of the above two methods can be done in polynomial time.

Model enumeration of a d-DNNF sentence is computed based on its equivalent sd-DNNF sentence. Model enumeration of a sentence in d-DNNF$_{\text{PI}}$ or d-DNNF$_{\text{NI}}$ should also be computed on its equivalent d-DNNF$_{\text{PNI}}$ sentence.

Definition 10: For a d-DNNF$_{\text{PNI}}$ sentence Σ rooted at Δ, IEnum(Δ) is defined as follows:
• IEnum(Δ) = $\{p = true\}$, if Δ is labeled with a positive literal p;
• IEnum(Δ) = $\{p = false\}$, if Δ is labeled with a negative literal $\neg p$;
• IEnum(Δ) = $\{p = true, p = false\}$, if Δ is labeled with an irrelevant variable p_I;
• IEnum(Δ) = $\{\{\}\}$, if Δ is labeled with *true*;
• IEnum(Δ) = $\{\}$, if Δ is labeled with *false*;
• IEnum(Δ) = $\cup_i$ IEnum(Δ), if Δ is labeled with *or*, where Δ_i are the children of Δ;
• IEnum(Δ) = $\{\cup_i \beta_i : \beta_i \in$IEnum($\Delta$)$\}$, if Δ is labeled with *and*, where Δ_i are the children of Δ.

Definition 10: gives a reasoning method of model enumeration on a d-DNNF$_{\text{PNI}}$ sentence.

For a d-DNNF$_{\text{PNI}}$ sentence Σ rooted at Δ and a clause C, clausal entailment checking $\Sigma| = S$ can be realized based on deciding the consistency of $\Sigma| \neg S$.

Definition 11: For a d-DNNF$_{\text{PNI}}$ sentence Σ rooted at Δ, S is a clause. EntailClause(Δ, S) is defined as follows:
• EntailClause(Δ, S) = *false*, if Δ is labeled with *false* or labeled with a literal which appears in S;
• EntailClause(Δ, S) = *true*, if Δ is labeled with other leaf node;
• EntailClause(Δ, S) = $\vee_i$ EntailClause(Δ, S), if Δ is labeled with *or*, where Δ_i are the children of Δ;
• EntailClause(Δ, S) = $\wedge_i$ EntailClause(Δ, S), if Δ is labeled with *and*, where Δ_i are the children of Δ.

The computing result of $\Sigma| = S$ is $\neg$EntailClause(Δ, S). Because the negative (resp. positive) literals are default in d-DNNF$_{\text{PI}}$ (resp. d-DNNF$_{\text{NI}}$) sentences, the values of some leaf nodes labeled with negative (resp. positive) literals cannot be changed to false if these literals are default. So, the reasoning method in Definition 11 is not appropriate for d-DNNF$_{\text{PI}}$ (resp. d-DNNF$_{\text{NI}}$), and clausal entailment checking on d-DNNF$_{\text{PI}}$ (resp. d-DNNF$_{\text{NI}}$) sentences should be done based on the corresponding d-DNNF$_{\text{PNI}}$ sentences. However, if S is a clause in which all literals are positive (resp. negative), the reasoning method in Definition 11 is also appropriate for d-DNNF$_{\text{PI}}$ (resp. d-DNNF$_{\text{NI}}$).

For a d-DNNF$_{\text{PNI}}$ sentence Σ rooted at Δ and a term T, implicant checking $T| = \Sigma$ can realized based on deciding the validity of $\Sigma|T$. For d-DNNF$_{\text{PI}}$ (resp. d-DNNF$_{\text{NI}}$) sentences, the negative literals (resp. positive literals) are default, so the conditioning on d-DNNF$_{\text{PI}}$ (resp. d-DNNF$_{\text{NI}}$) sentences should be done based on the corresponding d-DNNF$_{\text{PNI}}$ sentences. Then implicant checking on d-DNNF$_{\text{PI}}$ (resp. d-DNNF$_{\text{NI}}$) sentences should also be done based on the corresponding d-DNNF$_{\text{PNI}}$ sentences.

V. COMPRESSING d-DNNF WITH d-DNNF$_{\text{PI}}$ AND d-DNNF$_{\text{NI}}$

For some special problems in real world, such as blocks world planning problems and graph coloring problems, each satisfiable assignment for the SAT encodings of these problems must assign all variables, which means that there are no irrelevant variables in any satisfiable assignment. For these problems, we can only save positive literals or negative literals in each satisfiable assignment.

For other problems, there may exist a few irrelevant variables in their solutions. So, we can only save positive literals and irrelevant variables, or negative literals and irrelevant variables in each satisfiable assignment for these problems. That way, we can compile the formulae to d-DNNF$_{\text{PI}}$ or d-DNNF$_{\text{NI}}$, instead of d-DNNF.

Existing compiling algorithm of d-DNNF only compiles a formula to a sentence in d-DNNF. Since d-DNNF, d-DNNF$_{\text{PI}}$ and d-DNNF$_{\text{NI}}$ are three equivalent languages, compressing d-DNNF with d-DNNF$_{\text{PI}}$ and d-DNNF$_{\text{NI}}$ is a natural idea. Compiling a formula to a sentence in d-DNNF$_{\text{PI}}$ or d-DNNF$_{\text{NI}}$ does not reduce the online reasoning complexity with respect to d-DNNF. So we intend to compress d-DNNF with d-DNNF$_{\text{PI}}$ and d-DNNF$_{\text{NI}}$ in this paper.

Each sentence in d-DNNF can be converted to equivalence sentence in d-DNNF$_{\text{PI}}$ or d-DNNF$_{\text{NI}}$ in polynomial time. Given a sentence in d-DNNF, we can translate it to an equivalent sentence in d-DNNF$_{\text{PNI}}$, and select a more compact representation from sentences in d-DNNF$_{\text{PI}}$ and d-DNNF$_{\text{NI}}$. Above process can be completed in polynomial time based on Theorem 1.

VI. EXPERIMENTS

Compiling time and the size of results are two fundamental criterions for evaluating knowledge compilation methods [8]–[11], [21]. The sentences in d-DNNF$_{\text{PNI}}$, d-DNNF$_{\text{PI}}$ and d-DNNF$_{\text{NI}}$ can be translated from sd-DNNF sentences

TABLE 1. The comparison of compiling results for d-DNNF, sd-DNNF, d-DNNF$_{PNI}$, d-DNNF$_{PI}$ and d-DNNF$_{NI}$.

Benchmarks	d-DNNF	sd-DNNF	d-DNNF$_{PNI}$	d-DNNF$_{PI}$	d-DNNF$_{NI}$
bw(6)	5542	5542	5542	**720**	4841
flat(100)	2836740	2836740	2836740	**1601023**	2090449
grid(19)	1439266	1475975	1445575	**389363**	1397803
empr(31)	260996	261495	261124	**158452**	210477
stnt(9)	6261781	6267512	6262109	**6162405**	6237492
CBS(1000)	142873	170362	150155	84112	**83792**
RTI(500)	816260	1033010	926091	**664227**	665453
uf(100)	1532902	1978136	1804643	1510792	**1502257**
BMS(500)	**6272479**	8943069	8545768	7377897	7369718

in polytime, we do not design special compilers for new languages. So, we compare the sizes of sentences in different languages (d-DNNF, sd-DNNF, d-DNNF$_{PNI}$, d-DNNF$_{PI}$ and d-DNNF$_{NI}$) for same problems in this section. c2d is a classical and effective d-DNNF compiler, which can compile CNF formulae to equivalent d-DNNF sentences or sd-DNNF sentences. So we use c2d to compile problems in benchmarks to sentences in d-DNNF and sd-DNNF, then sentences in d-DNNF$_{PNI}$, d-DNNF$_{PI}$ and d-DNNF$_{NI}$ can be translated in polynomial time by Theorem 1. We also test the effect of compressing d-DNNF with d-DNNF$_{PI}$ and d-DNNF$_{NI}$. The sizes of compiling results for different problems are greatly different, even these problems are in the same kind. So we use average improvement of compressing effect to evaluate our compressing method for each kind of instances.

Experiments are conducted on a Windows desktop with a quad-core 3.30 GHz processor. Individual runs of c2d were limited to a 30-minute time-out. The benchmarks we used are: uniform random 3-SAT (*uf*), random 3-SAT Instances (*RTI*), backbone-minimal sub-instances (*BMS*), random-3-SAT instances with controlled backbone size (*CBS*), structured problems encoded as CNF (blocksworld, *bw*; and flat graph coloring, *flat*), and conformant planning problems converted to CNF (emptyroom, *empr*; *grid*; and sortnet, *stnt*).

In Table 1, *bw* and *flat* are two kinds of benchmarks, in which all solutions do not contain irrelevant variables. So the sizes of compiling results in d-DNNF, sd-DNNF and d-DNNF$_{PNI}$ are equal for above two kinds of benchmarks. Compared to sd-DNNF, the sizes of compiling results in d-DNNF$_{PNI}$ are a little less for all benchmarks except *bw* and *flat*.

For *bw*, *flat*, *grid*, *empr* and *stnt*, the sizes of compiling results in d-DNNF$_{PI}$ are least. For above kinds of benchmarks, they are encoded from corresponding actual problems, so there are some special structures in them. And there are more negative literals in the solutions of above benchmarks. So the sizes of compiling results in d-DNNF$_{PI}$ are significantly lee than the sizes of compiling results in other languages.

TABLE 2. The compression results for d-DNNF.

Benchmarks	d-DNNF	Compress		
		size	time(s)	avg_imp
bw(6)	5542	720	0.224	5.918
flat(100)	2836740	1601023	5.017	1.886
grid(19)	1439266	389363	13.912	3.702
empr(31)	260996	158452	0.653	1.663
stnt(9)	6261781	6162405	0.702	1.433
CBS(1000)	142873	78932	4.997	1.813
RTI(500)	816260	651059	4.273	1.524
uf(100)	1532902	1452186	4.819	1.398
BMS(500)	6272479	6004771	19.668	1.512

For *CBS*, *RTI*, *uf* and *BMS*, the sizes of compiling results in d-DNNF$_{PI}$ are approximately equal to the sizes of compiling results in d-DNNF$_{NI}$. The reason is that their structures do not influence the proportion of positive literals and negative literals in their solutions as they are generated randomly. Since there are many irrelevant variables in the solutions of some *BMS* benchmarks, the size of compiling results in d-DNNF is least.

Compressing results are listed in Table 2. Our compressing method is effective for all kinds of benchmarks. Specially, our compressing method averagely reduces the size of compiling result in d-DNNF to 1/5.918 of original size for each sentence in *bw*. Sentence with less size means online reasoning is more efficient. Overall, d-DNNF$_{NI}$ and d-DNNF$_{PI}$ have the same tractability as d-DNNF. So, we can completely replace the sentences in d-DNNF with the compressing result.

VII. CONCLUSION

Irrelevant variables are usually omitted in knowledge compilation languages, since they can be computed based on known positive literals and negative literals. We proposed three new knowledge compilation languages: NNF$_{PNI}$, NNF$_{PI}$ and NNF$_{NI}$, which are defined based on irrelevant variables in this paper. We proved that NNF, NNF$_{PNI}$, NNF$_{PI}$ and NNF$_{NI}$ can be translated to each other in polynomial time. We also defined three new knowledge compilation languages:

d-DNNF$_{\text{PNI}}$, d-DNNF$_{\text{PI}}$ and d-DNNF$_{\text{NI}}$, which are three meaningful subclasses of NNF$_{\text{PNI}}$, NNF$_{\text{PI}}$ and NNF$_{\text{NI}}$.

We designed new reasoning methods for d-DNNF$_{\text{PNI}}$, d-DNNF$_{\text{PI}}$ and d-DNNF$_{\text{NI}}$, mainly aiming at consistency check, validity check, clausal entailment check, implicant check, model counting, model enumeration, and conditioning transformation. Overall, d-DNNF$_{\text{PI}}$ and d-DNNF$_{\text{NI}}$ have the same tractability with d-DNNF. So, we compressed d-DNNF with d-DNNF$_{\text{PI}}$ and d-DNNF$_{\text{NI}}$. Experimental results show that the sizes of compiling results in d-DNNF$_{\text{PI}}$ or d-DNNF$_{\text{NI}}$ are less than the sizes of compiling results in d-DNNF, and our compressing method is effective for all kinds of benchmarks.

In the future, we will apply d-DNNF$_{\text{PNI}}$, d-DNNF$_{\text{PI}}$ and d-DNNF$_{\text{NI}}$ to relevant reasoning applications, design efficient knowledge compilers for d-DNNF$_{\text{PNI}}$, d-DNNF$_{\text{PI}}$ and d-DNNF$_{\text{NI}}$, and augment other knowledge compilation languages with irrelevant variables.

REFERENCES

[1] J. Newton and D. Verna, "A theoretical and numerical analysis of the worst-case size of reduced ordered binary decision diagrams," *ACM Trans. Comput. Logic*, vol. 20, no. 1, pp. 6:1-6:36, Feb. 2019.
[2] S. Jabbour, Y. Ma, B. Raddaoui, and L. Sais, "Quantifying conflicts in propositional logic through prime implicates," *Int. J. Approx. Reasoning*, vol. 89, pp. 27–40, Oct. 2017.
[3] Y. Salhi, "Approaches for enumerating all the essential prime implicants," in *Proc. 18th Int. Conf. Artif. Intell., Methodol., Syst., Appl. (AIMSA)*, Varna, Bulgaria, Sep. 2018, pp. 228–239.
[4] D. de Uña, G. Gange, P. Schachte, and P. J. Stuckey, "Compiling CP subproblems to MDDs and d-DNNFs," *Constraints*, vol. 24, no. 1, pp. 56–93, Jan. 2019.
[5] R. Hähnle, N. V. Murray, and E. Rosenthal, "Normal forms for knowledge compilation," in *Proc. 15th Int. Symp. Methodol. Intell. Syst. (ISMIS)*, Saratoga Springs, NY, USA, May 2005, pp. 304–313.
[6] R. Mateescu, R. Dechter, and R. Marinescu, "AND/OR multi-valued decision diagrams (AOMDDs) for graphical models," *J. Artif. Intell. Res.*, vol. 33, pp. 465–519, Dec. 2008.
[7] K. Pipatsrisawat and A. Darwiche, "New compilation languages based on structured decomposability," in *Proc. 23rd AAAI Conf. Artif. Intell. (AAAI)*, Chicago, IL, USA, Jul. 2008, pp. 517–522.
[8] A. Darwiche, "SDD: A new canonical representation of propositional knowledge bases," in *Proc. 22nd Int. Joint Conf. Artif. Intell. (IJCAI)*, Barcelona, Spain, Jul. 2011, pp. 819–826.
[9] M. Nishino, N. Yasuda, S. Minato, and M. Nagata, "Zero-suppressed sentential decision diagrams," in *Proc. 13th AAAI Conf. Artif. Intell. (AAAI)*, Phoenix, AZ, USA, Feb. 2016, pp. 819–826.
[10] Y. Lai, D. Y. Liu, and S. S. Wang, "Reduced ordered binary decision diagram with implied literals: A new knowledge compilation approach," *Knowl. Inf. Syst.*, vol. 35, no. 3, pp. 665–712, Jun. 2013.
[11] Y. Lai, D. Liu, and M. Yin, "New canonical representations by augmenting OBDDs with conjunctive decomposition," *J. Artif. Intell. Res.*, vol. 58, pp. 453–521, Mar. 2017.
[12] A. Darwiche and P. Marquis, "A knowledge compilation map," *J. Artif. Intell. Res.*, vol. 17, pp. 229–264, Sep. 2002.
[13] A. Darwiche, "On the tractable counting of theory models and its application to truth maintenance and belief revision," *J. Appl. Non-Classical Logics*, vol. 11, nos. 1–2, pp. 11–34, Jan. 2001.
[14] D. Shterionov and G. Janssens, "Crucial components in probabilistic inference pipelines," in *Proc. 30th Annu. ACM Symp. Appl. Comput. (SAC)*, Salamanca, Spain, Apr. 2015, pp. 1887–1889.
[15] D. Shterionov and G. Janssens, "Implementation and performance of probabilistic inference pipelines," in *Proc. 17th Int. Symp. Pract. Aspects Declarative Lang. (PADL)*, Portland, OR, USA, Jun. 2015, pp. 90–104.
[16] J. Vlasselaer, W. Meert, G. V. den Broeck, and L. De Raedt, "Exploiting local and repeated structure in dynamic Bayesian networks," *Artif. Intell.*, vol. 232, pp. 43–53, Mar. 2016.
[17] S. Siddiqi and J. B. Huang, "Probabilistic sequential diagnosis by compilation," in *Proc. Int. Symp. Artif. Intell. Math. (ISAIM)*, Fort Lauderdale, FL, USA, Jan. 2008, pp. 1–7.
[18] K. Pipatsrisawat and A. Darwiche, "Clone: Solving weighted MAX-SAT in a reduced search space," in *Proc. 20th Austral. Joint Conf. Adv. Artif. Intell.*, Gold Coast, QLD, Australia, Dec. 2007, pp. 223–233.
[19] A. Darwiche, "New advances in compiling CNF to decomposable negation normal form," in *Proc. 16th Eur. Conf. Artif. Intell. (ECAI)*, Valencia, Spain, Aug. 2004, pp. 328–332.
[20] C. Muise, S. McIlraith, J. C. Beck, and E. Hsu, "Dsharp: Fast d-DNNF compilation with sharpSAT," in *Proc. 25th Can. Conf. Adv. Artif. Intell.*, Toronto, ON, Canada, May 2012, pp. 356–361.
[21] P. Marquis and J. M. Lagniez, "An improved decision-DNNF compiler," in *Proc. 26th Int. Joint Conf. Artif. Intell. (IJCAI)*, Melbourne, VIC, Australia, Aug. 2017, pp. 667–673.

DANGDANG NIU received the M.Sc. and Ph.D. degrees in computer software and theory from the College of Computer Science and Technology, Jilin University, China, in 2015 and 2018, respectively. He is currently a Lecturer with the College of Information Engineering, Northwest A&F University, China. His research interests include automated reasoning and knowledge compilation.

LEI LIU received the M.Sc. degree in computer software and theory from the College of Computer Science and Technology, Jilin University, China, in 1985, where he is currently a Professor and the Ph.D. Supervisor. His research interests include software theory and technology.

SHUAI LÜ received the M.Sc. and Ph.D. degrees in computer software and theory from the College of Computer Science and Technology, Jilin University, China, in 2007 and 2010, respectively, where he is currently an Associate Professor. His research interests include artificial intelligence, machine learning, and automated reasoning.