Knowledge Compilation Meets Logical Separability

Junming Qiu[1, 2], **Wenqing Li**[1, 2], **Zhanhao Xiao**[3], **Quanlong Guan**[1, 2*], **Liangda Fang**[1, 4, 5*], **Zhao-Rong Lai**[1, 2], **Qian Dong**[1]

[1] Jinan University, Guangzhou 510632, China
[2] Guangdong Institute of Smart Education, Guangzhou 510632, China
[3] Sun Yat-sen University, Guangzhou 510006, China
[4] Pazhou Lab, Guangzhou 510330, China
[5] Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China
{2040697476jnu, eskii0706}@stu2020.jnu.edu.cn; xiaozhh9@mail.sysu.edu.cn; {gql, fangld, laizhr, dongq8}@jnu.edu.cn

Abstract

Knowledge compilation is an alternative solution to address demanding reasoning tasks with high complexity via converting knowledge bases into a suitable target language. Interestingly, the notion of logical separability, proposed by Levesque, offers a general explanation for the tractability of clausal entailment for two remarkable languages: decomposable negation normal form and prime implicates. It is interesting to explore what role logical separability on earth plays in problem tractability. In this paper, we apply the notion of logical separability in three reasoning problems within the context of propositional logic: satisfiability check (CO), clausal entailment check (CE) and model counting (CT), contributing to three corresponding polytime procedures. We provide three logical separability based properties: CO-logical separability, CE-logical separability and CT-logical separability. We then identify three novel normal forms: CO-LSNNF, CE-LSNNF and CT-LSNNF based on the above properties. Besides, we show that every normal form is the necessary and sufficient condition under which the corresponding procedure is correct. We finally integrate the above four normal forms into the knowledge compilation map.

Introduction

Knowledge compilation (KC) has been attracted interests in many areas of AI, for example, model-based diagnosis (Huang and Darwiche 2005; Mateescu, Dechter, and Marinescu 2008), explainable machine learning (Shih, Darwiche, and Choi 2019; Darwiche and Hirth 2020), probabilistic inference (Martires, Dries, and De Raedt 2019; Shih, Choi, and Darwiche 2019) and planning (Palacios et al. 2005; Huang 2006) and so on. As an alternative solution to address demanding reasoning tasks with high complexity, KC converts knowledge bases into a target language in which such reasoning tasks can be tractably accomplished.

Darwiche and Marquis (2002) proposed two criteria to evaluate target compilation languages: succinctness (the size of complied knowledge bases) and tractability (the supported polytime queries and transformations). In general, more succinct languages are less tractable and vice versa. One of primary purposes of KC is to design a suitable target language that achieves a good trade-off between succinctness and tractability for one or more specific reasoning tasks.

In the past decades, many prominent target compilation languages were developed, particularly, decomposable negation normal form (DNNF), the subset of negation normal form (NNF) satisfying the $\wedge$-decomposability property (Darwiche 2001a). The author also designed two polytime procedures for satisfiability check and clausal entailment check, which are correct for DNNF. In other words, DNNF supports polytime satisfiability check and clausal entailment check.

One question worth investigation is the intrinsic reason why DNNF supports such polytime queries. It actually can be answered by the notion of *logical separability* which was first proposed by Levesque (1998). Roughly speaking, a conjunction ϕ is logically separable, if the clausal entailment problem of a conjunction ϕ can be decomposed to entailment problems for every conjunct of ϕ. It is easily verified that any conjunction appearing in a DNNF-formula is logically separable due to the $\wedge$-decomposability property. In addition, prime implicate normal form (PI) supports polytime clausal entailment check as the conjunction of prime implicates is logically separable. Hence, logical separability offers a thorough explanation for why both DNNF and PI supports clausal entailment check.

It is interesting to explore what role logical separability on earth plays in problem tractability. In the paper (Levesque 1998), Levesque also defined a normal form, namely Levesque's normal form (LNF), s.t. the entailment check is tractable for a class of knowledge bases. However, after that, the idea of logical separability has hardly been applied to language design for other reasoning problems. It motivates us to take logical separability into account to design succinct target languages in which reasoning problems are tractably solved.

In this paper, we focus on three reasoning problems in the context of propositional logic: satisfiability check, clausal entailment check and model counting. For these three reasoning problems, we start by providing three polytime procedures $\mathcal{CO}$, $\mathcal{CE}$ and $\mathcal{CT}$, and then give definitions of logical separability based properties: CO-logical separability, CE-logical separability and CT-logical separability,

[*]Both are corresponding authors.

respectively. We also propose three novel normal forms: CO-LSNNF, CE-LSNNF and CT-LSNNF, each of which satisfies the above logical separability based properties, respectively, and show that every normal form is the necessary and sufficient condition under which the corresponding procedure is correct. In addition, CO-LSNNF, CE-LSNNF and CT-LSNNF are complete languages.

- For CO-LSNNF, we provide two interesting theoretical results: (1) any language supports polytime satisfiability check iff it is polynomially translatable into CO-LSNNF; and (2) CO-LSNNF is equally succinct to NNF.
- For CE-LSNNF, we show that if a language is polynomially translatable into CE-LSNNF, then it supports polytime clausal entailment check; and make a comparison to LNF which is a strictly less succinct than CE-LSNNF.
- For CT-LSNNF, we prove that (1) if a language is polynomially translatable into CT-LSNNF, then it supports polytime model counting, and (2) CT-LSNNF is polynomially equivalent to d-DNNF.

Furthermore, we analyze the computational complexity of the membership problems of the three logical separability-based normal forms. Finally, we analyze the relative succinctness of CO-LSNNF, CE-LSNNF, CT-LSNNF and LNNF compared to the languages considered in (Darwiche and Marquis 2002; Fargier and Marquis 2014) and investigate which of the queries and transformations can be accomplished in polytime for them.

Preliminaries

The NNF Languages

Throughout this paper, we fix a finite set $\mathbf{X}$ of variables. A *literal* is a variable (positive literal) or a negated one (negative literal). For a positive (resp. negative) literal x (resp. $\neg x$), its complementary literal is $\neg x$ (resp. x). Let $\mathbf{Y} \subseteq \mathbf{X}$. A $\mathbf{Y}$-literal is a literal x or $\neg x$ where $x \in \mathbf{Y}$. A *term* (resp. *clause*) is $\top$, $\bot$, or a conjunction (resp. disjunction) of literals. We say a term (resp. clause) is *non-trivial* if every variable appears at most once.

Throughout this paper, we consider a wide range of complete propositional languages, all of which are the subsets of negation normal form (NNF). A NNF-formula is a rooted, directed acyclic graph (DAG), of which each leaf node is labeled by $\top$, $\bot$, or literals; and each internal node is labeled by $\wedge$ (conjunction) or $\vee$ (disjunction). We use a lower-case Greek letter (e.g. α, β) to denote a propositional formula. We use $\mathrm{Var}(\alpha)$ (resp. $\mathrm{Lit}(\alpha)$) to denote the set of variables (resp. literals) appearing in α. We use $|\alpha|$ to denote the size of α (*i.e.* the number of its DAG edges).

A $\mathbf{Y}$-*interpretation* ω is a set of $\mathbf{Y}$-literals s.t. each variable of $\mathbf{Y}$ appears exactly once. Let $\mathrm{Var}(\alpha) \subseteq \mathbf{Y}$. We use $\omega \models \alpha$ to denote ω satisfies α, which is defined as usual. A $\mathbf{Y}$-*model* of α is a $\mathbf{Y}$-interpretation satisfying α. We use $\mathrm{Mod}_{\mathbf{Y}}(\alpha)$ to denote the set of $\mathbf{Y}$-models of α. For simplicity, in the case $\mathbf{Y} = \mathbf{X}$, we omit the subscript $\mathbf{X}$, and $\mathrm{Mod}(\alpha)$ denotes the set of $\mathbf{X}$-models of α. We say α is *satisfiable*, if $\mathrm{Mod}(\alpha) \neq \emptyset$; otherwise, it is *unsatisfiable*.

As mentioned in (Darwiche and Marquis 2002), a language qualifies as a target language if it supports polytime clausal entailment check. NNF is not a desired target language as it does not support such a query unless $\mathsf{P} = \mathsf{NP}$. But many of its subsets, with one or more restrictions, do.

*Conjunctive normal form (*CNF*)* is the conjunction of non-trivial clauses while its dual, *disjunctive normal form (*DNF*)*, is the disjunction of non-trivial terms. A *prime implicate* c of α is an implicate of α and there is no implicate $c' \neq c$ *s.t.* $\alpha \models c'$ and $c' \models c$. *Prime implicates (*PI*)* is the subset of CNF where each formula α is a conjunction of all of the prime implicates of α. Its dual, *prime implicants (*IP*)*, can be similarly defined.

Decomposable NNF *(*DNNF*)* (Darwiche 2001a) is the subset of NNF satisfying $\wedge$-decomposability, requiring the sets of variables of the children of each $\wedge$-node in a formula to be pairwise disjoint. *Deterministic DNNF (*d-DNNF*)* (Darwiche 2001b) is the subset of DNNF satisfying determinism, requiring the children of each $\vee$-node in a formula to be pairwise logically contradictory.

KROM (Krom 1970) is the subset of CNF in which each clause contains at most two literals. HORN (Horn 1951) is the subset of CNF in which each clause contains at most a positive literal. K/H is the union of KROM and HORN. *Renamable-Horn* (renH) is the subset of all CNF-formulas α for which there is a subset $\mathbf{Y}$ of $\mathrm{Var}(\alpha)$ s.t. the formula obtained by substituting in α every $\mathbf{Y}$-literal l by its complement $\bar{l}$ is a HORN-formula. Fargier and Marquis (2014) applied disjunctive closure principle to KROM, HORN, K/H and renH and obtained KROM[$\vee$], HORN[$\vee$], K/H[$\vee$] and renH[$\vee$], respectively.

We remark that the above normal forms are not only defined in purely syntactic style but also based on semantics. For example, CNF and DNF are syntactic normal forms. On the other hand, PI and d-DNNF are based on clausal entailment and satisfiability, respectively and hence being semantic normal forms.

Succinctness and Polynomial Translations

We now consider two notions of translations on two subsets of NNF: *succinctness* and *polynomial-translation*.

Definition 1. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be subsets of NNF. We say

- $\mathcal{L}_1$ is at least as succinct as $\mathcal{L}_2$, denoted $\mathcal{L}_1 \leq_s \mathcal{L}_2$, if there is a polynomial p s.t. for every formula $\alpha \in \mathcal{L}_2$, there is an equivalent formula $\beta \in \mathcal{L}_1$ s.t. $|\beta| \leq p(|\alpha|)$.
- $\mathcal{L}_2$ is polynomially translatable into $\mathcal{L}_1$, denoted $\mathcal{L}_1 \leq_p \mathcal{L}_2$, if there exists a (deterministic) polynomial-time algorithm f s.t. for every formula $\alpha \in \mathcal{L}_2$, we have an equivalent formula $f(\alpha) \in \mathcal{L}_1$.

Succinctness only considers polynomial-space translations, that is, the size of the $\mathcal{L}_1$-formula β equivalent to α is required to be polynomial in the size of $\mathcal{L}_2$-formula α. Polynomial-translation is a more strict relation than succinctness, requiring polynomial-time translations, that is, any $\mathcal{L}_2$-formula can be tractably transformed into an equivalent $\mathcal{L}_1$-formula. Furthermore, we have $\mathcal{L}_1 \leq_p \mathcal{L}_2$ implies that $\mathcal{L}_1 \leq_s \mathcal{L}_2$ (Fargier and Marquis 2014).

The two relations $\leq_s$ and $\leq_p$ are clearly reflexive and transitive, *i.e.* pre-orders over subsets of NNF. The notation

$\sim_s$ denotes the symmetric part of $\leq_s$, that is, $\mathcal{L}_1 \sim_s \mathcal{L}_2$ iff $\mathcal{L}_1 \leq_s \mathcal{L}_2$ and $\mathcal{L}_2 \leq_s \mathcal{L}_1$. On the other hand, $<_s$ denotes the asymmetric part of $\leq_s$, that is, $\mathcal{L}_1 <_s \mathcal{L}_2$ iff $\mathcal{L}_1 \leq_s \mathcal{L}_2$ and $\mathcal{L}_2 \not\leq_s \mathcal{L}_1$. In the following, $\mathcal{L}_1 \not\leq_s^* \mathcal{L}_2$ means that $\mathcal{L}_1 \not\leq_s \mathcal{L}_2$ unless the polynomial hierarchy PH collapses. The notations $\sim_p$, $<_p$ and $\not\leq_p^*$ can be similarly defined.

Queries and Transformations

As pointed out in (Darwiche and Marquis 2002), another two crucial criteria for evaluating normal forms is the set of query and transformation tasks supported in polytime. To differentiate query (or transformation) tasks and properties, we use letters in typewriter type font for the former, and letters in the boldface font for the latter.

A *query task* for a language $\mathcal{L}$ takes as input one or more $\mathcal{L}$-formulas and possibly clauses and terms, and outputs a Boolean value, a natural number, or a set of interpretations. We consider the following query tasks:

- Satisfiability (`CO`): check if $\alpha \not\equiv \bot$.
- Validity (`VA`): check if $\alpha \equiv \top$.
- Casual entailment (`CE`): check if $\alpha \models c$ where c is a clause.
- Term implication (`IM`): check if $t \models \alpha$ where t is a term.
- Sentential entailment (`SE`): check if $\alpha \models \beta$.
- Equivalence (`EQ`): check if $\alpha \equiv \beta$.
- Model counting (`CT`): compute the size of $\mathrm{Mod}(\alpha)$.
- Model enumeration (`ME`): list every member of $\mathrm{Mod}(\alpha)$.

A *transformation task* for $\mathcal{L}$ takes as input one or a set of $\mathcal{L}$-formulas, possibly a term and a set of variables, and returns an appropriate $\mathcal{L}$-formula. We consider the following transformation tasks:

- Conditioning (`CD`): generate $\alpha|t$ where t is a satisfiable term;
- Forgetting (`FO`): generate $\exists \mathbf{Y}.\alpha$ where $\mathbf{Y} \subseteq \mathbf{X}$.
- Singleton forgetting (`SFO`): generate $\exists\{x\}.\alpha$ where $x \in \mathbf{X}$.
- Conjunction ($\wedge$`C`): generate $\alpha_1 \wedge \cdots \wedge \alpha_n$.
- Disjunction ($\vee$`C`): generate $\alpha_1 \vee \cdots \vee \alpha_n$.
- Bounded conjunction ($\wedge$`BC`): generate $\alpha \wedge \beta$.
- Bounded disjunction ($\vee$`BC`): generate $\alpha \vee \beta$.
- Negation ($\neg$`C`): generate $\neg\alpha$.

Specifically, conditioning a formula α on a satisfiable term t, denoted by $\alpha|t$, is the result of replacing each occurrence of a variable x in α by $\top$ (resp. $\bot$), if x (resp. $\neg x$) is a positive (resp. negative) literal of t. Forgetting $\mathbf{Y}$ from α, denoted as $\exists \mathbf{Y}.\alpha$, is the strongest consequence that does not contains any variables of $\mathbf{Y}$, that is, for any `NNF`-formula β s.t. Var$(\beta) \cap \mathbf{Y} = \emptyset$, $\alpha \models \beta$ iff $\exists \mathbf{Y}.\alpha \models \beta$. We say a language $\mathcal{L}$ satisfies the query (resp. transformation) property $\mathbf{Q}$ (resp. $\mathbf{T}$), iff there is a polytime procedure for the query (resp. transformation) task `Q` (resp. `T`) for $\mathcal{L}$. For example, `DNF` supports **CO** since the `CO` task can be performed under `DNF` in polytime (Darwiche and Marquis 2002).

Satisfiability

We first present a procedure for satisfiability check on `NNF`-formulas, which was proposed by Darwiche (2001a).

Definition 2. The procedure $\mathcal{CO}(\alpha)$ is recursively defined as:

- $\mathcal{CO}(\top) = 1, \mathcal{CO}(\bot) = 0$ and $\mathcal{CO}(l) = 1$
- $\mathcal{CO}(\beta_1 \wedge \cdots \wedge \beta_n) = \min\limits_{1 \leq i \leq n} \{\mathcal{CO}(\beta_i)\}$
- $\mathcal{CO}(\beta_1 \vee \cdots \vee \beta_n) = \max\limits_{1 \leq i \leq n} \{\mathcal{CO}(\beta_i)\}$

The above procedure works in a recursive way. In the base case, the Boolean constant $\top$ and a literal l are satisfiable while $\bot$ is unsatisfiable. In the induction case, the satisfiability problem of an $\vee$-node (or $\wedge$-node) is reduced to the same problem of its subformula β_i. If some β_i of $\vee$-node α is satisfiable, then α is satisfiable. Similarly, if every β_i of $\wedge$-node α is satisfiable, then α is satisfiable.

Definition 3. Let $\mathcal{L}$ be a language, and f a procedure that takes an `NNF`-formula α as input, and that returns a Boolean value. We say

- f is `CO`-sound for $\mathcal{L}$ iff for every $\mathcal{L}$-formula α, $f(\alpha) = 1$ only if `CO`$(\alpha) = 1$;
- f is `CO`-complete for $\mathcal{L}$ iff for every $\mathcal{L}$-formula α, `CO`$(\alpha) = 1$ only if $f(\alpha) = 1$.

The procedure $\mathcal{CO}$ is a `CO`-complete algorithm for any `NNF`-formula α and it can be evaluated in linear time in $|\alpha|$.

Theorem 1. (Darwiche 2001a) The procedure $\mathcal{CO}$ is `CO`-complete for `NNF` with the time complexity $O(|\alpha|)$.

However, the procedure $\mathcal{CO}$ is unsound for `NNF`-formula. Let us consider the unsatisfiable formula $\alpha = x \wedge \neg x$. Clearly, $\mathcal{CO}(x) = \mathcal{CO}(\neg x) = 1$. It follows that $\mathcal{CO}(\alpha) = 1$ which contradicts the unsatisfiability of α.

The reason why the procedure $\mathcal{CO}$ cannot serve as a sound algorithm is that $\mathcal{CO}$ is a simple evaluation-based algorithm and it does not consider implicit logical contraditions occurring in some $\wedge$-nodes of an `NNF`-formula. Formally, we say an $\wedge$-node $\alpha = \beta_1 \wedge \cdots \wedge \beta_n$ contains an implicit logical contradiction, if α is unsatisfiable and each β_i is satisfiable for $1 \leq i \leq n$. In this case, $\mathcal{CO}(\alpha) = 1$ as $\mathcal{CO}(\beta_i) = 1$ for $1 \leq i \leq n$, and hence deriving an incorrect result.

We hereafter provide a necessary and sufficient condition on `NNF` for which the $\mathcal{CO}$ procedure is complete and sound.

Definition 4. An `NNF`-formula α is `CO`-logically separable (`CO`$_{ls}$) iff

- α is $\top$, $\bot$, or a literal l; or
- α is an $\wedge$-node $\beta_1 \wedge \cdots \wedge \beta_n$, and if α is unsatisfiable, then β_i is unsatisfiable and `CO`$_{ls}$ for some $1 \leq i \leq n$; or
- α is an $\vee$-node $\beta_1 \vee \cdots \vee \beta_n$, and if α is unsatisfiable, then β_i is unsatisfiable and `CO`$_{ls}$ for every $1 \leq i \leq n$.

We use `CO-LSNNF` to denote the subset of `NNF` in which any formula satisfies `CO`-logical separability property.

For instance, the formula $x \wedge \neg x \wedge \bot$ is `CO`-logically separable as the logical contradiction occurs explicitly.

To avoid the implicit logical contraditions in an unsatisfiable $\wedge$-node $\beta_1 \wedge \cdots \wedge \beta_n$, Definition 4 requires some conjuncts β_i to be unsatisfiable and `CO`-logically separable.

Let us consider the formula $\alpha = ((x \wedge \neg x) \vee y) \wedge \bot$. The logical contradiction hidden in the inner $\wedge$-node $x \wedge \neg x$ does not impede the correctness of the procedure $\mathcal{CO}$ due to the occurrence of $\bot$ in the outermost $\wedge$-node of α. In a similar way, each disjunct of an unsatisfiable $\vee$-node is unsatisfiable and CO-logically separable. Hence, we obtain the soundness of the procedure $\mathcal{CO}$.

Theorem 2. A language $\mathcal{L}$ is CO-LSNNF iff the procedure $\mathcal{CO}$ is CO-complete and CO-sound for $\mathcal{L}$.

Proof. We prove by induction on α. We here only verify the case where α is an $\wedge$-node $\beta_1 \wedge \cdots \wedge \beta_n$.

($\Rightarrow$): It directly follows from Theorem 1 that the procedure $\mathcal{CO}$ is complete for CO-LSNNF. It remains to verify the soundness property of the procedure $\mathcal{CO}$ for CO-LSNNF. Let α be an arbitrary CO-LSNNF formula. Assume that $\mathtt{CO}(\alpha) = 0$. We hereafter prove that $\mathcal{CO}(\alpha) = 0$. By CO-logical separability property, there is some β_i that is unsatisfiable and $\mathtt{CO}_{ls}$. By the induction hypothesis, we have that $\mathcal{CO}(\beta_i) = 0$. It follows from the procedure $\mathcal{CO}$ that $\mathcal{CO}(\alpha) = 0$.

($\Leftarrow$): Assume that the language $\mathcal{L}$ is such that for any $\mathcal{L}$-formula α, $\mathtt{CO}(\alpha) = 1$ iff $\mathcal{CO}(\alpha) = 1$. In case where α is satisfiable, by Definition 4, it holds that $\alpha \in$ CO-LSNNF. In case where α is unsatisfiable. By the assumption above, we get that $\mathcal{CO}(\alpha) = 0$. According to the procedure $\mathcal{CO}$, we get that $\mathcal{CO}(\beta_i) = 0$ for some i. Since the procedure $\mathcal{CO}$ is complete, $\mathtt{CO}(\beta_i) = 0$. It follows that $\beta_i \in \mathcal{L}$. By the induction hypothesis, β_i is CO-logically separable. By Definition 4, α is also CO-logically separable. $\square$

We observe that all normal forms, summarized in (Darwiche and Marquis 2002), supporting **CO** include PI, DNNF, OBDD and so on. Interestingly, they are either PI or a subset of DNNF. In addition, we observe that every formula of PI or DNNF satisfies CO-logical separability.

Proposition 1. *The languages* DNNF *and* PI *are subsets of* CO-LSNNF.

On the one side, $\wedge$-decomposability forbids simultaneous occurrence of the same variable in distinct conjuncts of any $\wedge$-node. Hence, no implicit logical contradiction occurs in $\wedge$-nodes of a DNNF-formula. On the other side, PI requires that any implicate of α is derived by a clause c of α and that no implicate $c' \neq c$ of α such that $c' \models c$. The Boolean constant $\bot$ is a disjunction of an empty set of literals. If α implies $\bot$, then it must be $\bot$ itself. The logical contradiction therefore appears explicitly in any unsatisfiable PI-formula.

Someone may wonder if the two properties polytime satisfiability check and CO-logical separability are essentially equivalent? The four normal forms: KROM[$\vee$], HORN[$\vee$], renH[$\vee$] and K/H[$\vee$], which supports polytime satisfiability check (Fargier and Marquis 2014), are counterexamples for the above question. As mentioned before, the formula $x \wedge \neg x$ is not CO-logically separable. Clearly, it is a formula of the above four normal forms.

The story does not come to the end. We connect these two properties via polytime translation.

Theorem 3. *A language $\mathcal{L}$ satisfies* **CO** *iff it is polynomially translatable into* CO-LSNNF.

Proof. ($\Rightarrow$): Assume that $\mathcal{L}$ supports polytime satisfiability check. We design an algorithm transforming any $\mathcal{L}$-formula α into an equivalent one that is CO-logically separable. If it is satsifiable, then it is CO-logically separable; otherwise, the algorithm returns $\bot$ which is CO-logically separable. Since the satisfiability of $\mathcal{L}$-formula α can be achieved in time polynominal in $|\alpha|$, the above algorithm is also polytime.

($\Leftarrow$): Let f be the polytime algorithm that transforms any $\mathcal{L}$-formula α into an equivalent one that is CO-logically separable. By the completeness and soundness of the procedure $\mathcal{CO}$ for CO-LSNNF, α is satisfiable iff $\mathcal{CO}(f(\alpha)) = 1$. This, together with the fact that f and $\mathcal{CO}$ are polytime, imply that $\mathcal{L}$ satisfies **CO**. $\square$

In other words, $\mathcal{L}$ supports polytime satisfiability check iff the implicit logically contraditions in any $\mathcal{L}$-formula α that are witnesses to refute the satisfiability of α can be efficiently discovered.

We close this section by comparing CO-LSNNF and NNF in terms of succinctness and polynomial-translation.

Proposition 2.

- NNF $\sim_s$ CO-LSNNF
- NNF $\leq_p$ CO-LSNNF *and* CO-LSNNF $\not\leq_p^*$ NNF.

As mentioned in (Fargier and Marquis 2014), the inclusion $\leq_p \subset \leq_s$ holds. It means that there are two languages $\mathcal{L}_1$ and $\mathcal{L}_2$ such that $\mathcal{L}_1 \leq_s \mathcal{L}_2$ holds but $\mathcal{L}_1 \leq_p \mathcal{L}_2$ does not. However, they do not provide two languages confirming the inclusion $\leq_p \subset \leq_s$. We have surveyed the existing literature regarding succinctness among various propositional representations (Darwiche and Marquis 2002; Wachter and Haenni 2006; Pipatsrisawat and Darwiche 2008; Fargier and Marquis 2009; Darwiche 2011; Fargier and Marquis 2014; Berre et al. 2018; Čepek and Chromý 2020). We find that in these literature, for every two languages $\mathcal{L}_1$ and $\mathcal{L}_2$, if the succinctness relation $\mathcal{L}_1 \leq_s \mathcal{L}_2$ holds, then the polynomial-translation relation $\mathcal{L}_1 \leq_p \mathcal{L}_2$ also holds. To the best of our knowledge, this paper is the first one to provide the pair of languages which is a witness to the inclusion $\leq_p \subset \leq_s$.

A classical knowledge compilation result in (Selman and Kautz 1996) states that unless PH collapses, there does not exist a class $\mathcal{L}$ of formulas s.t. every NNF-formula has a polysize equivalent $\mathcal{L}$-formula, and $\mathcal{L}$ supports polytime clausal entailment. It is well-known that both CE and CO tasks for NNF are reasoning problems with high computational complexity, which are coNP-complete and NP-complete, respectively. In this paper, we show that CO-LSNNF is the language that supports polytime satisfiability check and that is equivalent succinct to NNF, proving the similar result in (Selman and Kautz 1996) for satisfiability check does not hold.

Finally, determining if an NNF-formula is CO-logically separable is the same as hard as the satisfiability problem.

Proposition 3. *Deciding if an* NNF*-formula is in* CO-LSNNF *is* NP*-complete.*

Proof. (Lower bound): By CO-logically separability and Theorem 2, we get that for any NNF formula α, α is satisfiable iff α is in CO-LSNNF and $\mathcal{CO}(\alpha) = 1$. Due to the fact that the satisfiability problem of any NNF-formula is NP-complete, we can draw a conclusion that deciding if an NNF-formula is in CO-LSNNF is NP-hard.

(Upper bound): It is easy to verify that for any NNF formula α, α is in CO-LSNNF iff α is satsifiable or $\mathcal{CO}(\alpha) = 0$. The satisfiability problem is in NP and so is the CO-LSNNF membership problem. □

Clausal Entailment

As mentioned in (Darwiche and Marquis 2002), a language qualifies as a target language, if it permits polytime clausal entailment check. In this section, we consider logical separability in the query clausal entailment.

The Procedure for Clausal Entailment

Definition 5. Let α be an NNF-formula and c a non-trivial clause. The procedure $\mathcal{CE}(\alpha, c)$ is recursively defined as:

- $\mathcal{CE}(\top, c) = \begin{cases} 1, & \text{if } c = \top \\ 0, & \text{otherwise} \end{cases}$
- $\mathcal{CE}(\bot, c) = 1, \mathcal{CE}(l, \top) = 1$ and $\mathcal{CE}(l, \bot) = 0$
- $\mathcal{CE}(l, l_1 \vee \cdots \vee l_m) = \begin{cases} 1, & \text{if } l = l_i \text{ for some } i \\ 0, & \text{otherwise} \end{cases}$
- $\mathcal{CE}(\beta_1 \wedge \cdots \wedge \beta_n, c) = \max\limits_{1 \leq i \leq n} \{\mathcal{CE}(\beta_i, c)\}$
- $\mathcal{CE}(\beta_1 \vee \cdots \vee \beta_n, c) = \min\limits_{1 \leq i \leq n} \{\mathcal{CE}(\beta_i, c)\}$

The above procedure is a recursive algorithm. In the case that α is a Boolean constant $\top$, the procedure returns 1 if the clause c is also $\top$, and returns 0 otherwise. In the case that α is $\bot$, the procedure always returns 1 since an unsatisfiable KB derives any consequence. In the case that α is a literal l, then the procedure returns 1 if l is a literal of the clause c, or c is $\top$ since l entails a clause including l and $\top$ is a consequence of any formula. The clausal entailment problem of an $\wedge$-node $\beta_1 \wedge \cdots \wedge \beta_n$ and a clause c can be reduced to the same subproblem of β_i and c. If one of the β_i's entails c, then their conjunction does. The case that α is a $\vee$-node is similarly handled.

Darwiche (2001a) proposed a polytime clausal entailment procedure $\mathcal{CE}'$ via the procedure $\mathcal{CO}$ and conditioning. The definition of $\mathcal{CE}'$ is as follows: $\mathcal{CE}'(\alpha, c) = 1$ iff $\mathcal{CO}(\alpha|t) = 0$ where t is equivalent to $\neg c$. In fact, the two procedures $\mathcal{CE}$ and $\mathcal{CE}'$ are equivalent, that is, they return the same result for every formula α and every non-trivial clause c.

Proposition 4. Let α be an NNF-formula and c a non-trivial clause. Then, $\mathcal{CE}(\alpha, c) = \mathcal{CE}'(\alpha, c)$.

The major advantage of the procedure $\mathcal{CE}$ over $\mathcal{CE}'$ is that the former directly solves the clausal entailment problem while the latter resorts to two procedures for satisfiability check and conditioning.

Definition 6. Let $\mathcal{L}$ be a language and f be a procedure that takes an NNF-formula α and a non-trivial clause c as input, and that returns a Boolean value. We say

- f is CE-sound for $\mathcal{L}$ iff for every $\mathcal{L}$-formula α and every non-trivial clause c, $f(\alpha, c) = 1$ only if $\text{CE}(\alpha, c) = 1$;
- f is CE-complete for $\mathcal{L}$ iff for every $\mathcal{L}$-formula α and every non-trivial clause c, $\text{CE}(\alpha,c) = 1$ only if $f(\alpha,c) = 1$.

The procedure $\mathcal{CE}$ is a sound algorithm for clausal entailment and takes polytime in the size of α and c.

Theorem 4. The procedure $\mathcal{CE}$ is CE-sound for NNF with the time complexity $O(|\alpha| \cdot |c|)$.

However, it is not guaranteed that the procedure $\mathcal{CE}$ is complete if α is an arbitrary NNF-formula. Let us consider the formula $\alpha = (\neg x \vee y) \wedge (\neg y \vee z)$ and the clause $c = \neg x \vee z$. It is easily verified that $\mathcal{CE}(\neg x, c) = 1$, $\mathcal{CE}(y, c) = 0$, $\mathcal{CE}(\neg y, c) = 0$ and $\mathcal{CE}(z, c) = 1$. It follows that $\mathcal{CE}(\neg x \vee y, c) = 0$ and $\mathcal{CE}(\neg y \vee z, c) = 0$. Finally, $\mathcal{CE}(\alpha, c) = 0$. On the contrary, $\text{CE}(\alpha, c) = 1$ as $\alpha \models c$.

The reason that the procedure $\mathcal{CE}$ cannot give a complete answer for clausal entailment is similar to why the $\mathcal{CO}$ does not work perfectly for satisfiability check. The procedure $\mathcal{CE}$ only decomposes $\wedge$-nodes of a formula α in a simple way and does not reasoning about implicit logical implicate of these $\wedge$-nodes. We say an $\wedge$-node $\alpha = \beta_1 \wedge \cdots \wedge \beta_n$ contains an implicit logical implicate, if there is a clause c s.t. $\alpha \models c$ and $\beta_i \not\models c$ for every $1 \leq i \leq n$.

By conjoining α with the clause $\neg x \vee z$, the new formula $\alpha' = (\neg x \vee y) \wedge (\neg y \vee z) \wedge (\neg x \vee z)$ contains no implicit logical implicates. We now examine the procedure $\mathcal{CE}$ for the formula α', which is equivalent to the previous one α. It is easily verified that $\mathcal{CE}(\neg x \vee z, c) = 1$, and hence $\mathcal{CE}(\alpha', c) = 1$. The procedure $\mathcal{CE}(\alpha', c)$ generates a result in accordance with $\text{CE}(\alpha', c)$.

Based on the above observations, we hereafter give a normal form that is the sufficient and necessary condition of making the procedure $\mathcal{CE}$ not only sound but also complete.

Definition 7. Let c be a non-trivial clause. An NNF-formula α is CE-logically separable (CE_{ls}) w.r.t. c iff

- α is $\top$, $\bot$, or a literal l; or
- α is an $\wedge$-node $\beta_1 \wedge \cdots \wedge \beta_n$, and if $\alpha \models c$, then there is a conjunct β_i s.t. $\beta_i \models c$ and β_i is CE_{ls} w.r.t. c; or
- α is an $\vee$-node $\beta_1 \vee \cdots \vee \beta_n$, and if $\alpha \models c$, then for every disjunct β_i, we have $\beta_i \models c$ and β_i is CE_{ls} w.r.t. c.

The Boolean constants and a literal l are CE_{ls} w.r.t. any non-trivial clause c. Suppose that α is an $\wedge$-node $\beta_1 \wedge \cdots \wedge \beta_n$ which entails a clause c. The clause c may be an implicit logical implicate of α. To avoid them, Definition 7 requires some of its conjunct to entail c and to be CE-logically separable w.r.t. c. In the similar way, if a $\vee$-node entails clause c, then each of its disjuncts also entails c and is CE-logically separable w.r.t. c.

We say an NNF-formula satisfies general CE-logically separable (general CE_{ls}), if it is CE_{ls} w.r.t. every non-trivial clause. We use CE-LSNNF to denote the subset of NNF for which every formula satisfies general CE_{ls}. General CE-logical separability eliminates any implicit implicate of some $\wedge$-nodes so as to gain the completeness of the procedure $\mathcal{CE}$ for every formula and every non-trivial clause.

Theorem 5. *A language $\mathcal{L}$ is* CE-LSNNF *iff the procedure $\mathcal{CE}$ is* CE-*complete and* CE-*sound for $\mathcal{L}$.*

Proof. We prove by induction on α. We here only verify the case where α is an $\wedge$-node $\beta_1 \wedge \cdots \wedge \beta_n$.

($\Rightarrow$): By Theorem 4, the procedure $\mathcal{CE}$ is CE-sound for CE-LSNNF. It remains to verify the completeness property of the procedure $\mathcal{CE}$ for CE-LSNNF. Let α be an arbitrary CE-LSNNF-formula and c a non-trivial clause. Assume that $\alpha \models c$. We hereafter prove that $\mathcal{CE}(\alpha, c) = 1$. By CE-logical separability property, we have that $\beta_i \models c$ and β_i is $\mathtt{CE}_{ls}$ w.r.t. c for some i. By the induction hypothesis, it holds that $\mathcal{CE}(\beta_i, c) = 1$. It follows from the procedure $\mathcal{CE}$ that $\mathcal{CE}(\alpha, c) = 1$.

($\Leftarrow$): Assume that the language $\mathcal{L}$ is such that for every $\mathcal{L}$-formula α and every non-trivial clause c, $\mathtt{CE}(\alpha, c) = 1$ iff $\mathcal{CE}(\alpha, c) = 1$. We prove that every $\mathcal{L}$-formula α is CE-logically separable w.r.t. every non-trivial clause c. In case where $\alpha \not\models c$, by Definition 7, it holds that α is $\mathtt{CE}_{ls}$ w.r.t. c. In case where $\alpha \models c$. By the assumption above, we get that $\mathcal{CE}(\alpha, c) = 1$. According to the procedure $\mathcal{CE}$, we get that $\mathcal{CE}(\beta_i, c) = 1$ for some i. Since the procedure $\mathcal{CE}$ is sound for any NNF-formula, we have that $\mathtt{CE}(\beta_i, c) = 1$. It follows that $\beta_i \in \mathcal{L}$. By the induction hypothesis, β_i is CE-logically separable w.r.t. c. By Definition 7, α is $\mathtt{CE}_{ls}$ w.r.t. c. □

We observe that $\wedge$-decomposability and prime implicates are special cases of general CE-logical separability.

Proposition 5. *DNNF and PI are subsets of CE-LSNNF.*

We elaborate the intuition behind $\wedge$-decomposability and prime implicant via the inference rule: resolution. Resolution is used to find a refutation proof of CNF-formulas (Davis and Putnam 1960). Levesque (1998) used resolution to discover implicit implicates hidden in $\wedge$-nodes. The conjunction of $x \vee c_1$ and $\neg x \vee c_2$ deduces their resolvant $c_1 \vee c_2$. This deduction also holds if the two clauses c_1 and c_2 are replaced by two arbitrary formulas β_1 and β_2 respectively.

On the one side, $\wedge$-decomposability precludes the case that a variable x simultaneously occurs in different conjuncts of an $\wedge$-node α. This causes no occurrence of implicit implicate within any $\wedge$-node. On the other side, PI-formula α requires every resolution among every two clauses of α to be entailed by a clause within α. Hence, no implicit implicate occurs via making advantage of resolution.

It is easily verified that every language $\mathcal{L}$ that is polynomially translatable into CE-LSNNF also satisfies **CE**.

Theorem 6. *If a language $\mathcal{L}$ is polynomially translatable into CE-LSNNF, then $\mathcal{L}$ satisfies* **CE**.

But the opposite direction remains open. Considering the four disjunctive closure based languages, none of these languages is at least as succinct as CE-LSNNF. Any KROM-formula can be converted into a PI-formula in polytime (Marquis 2000). In addition, PI is a subset of CE-LSNNF and CE-LSNNF supports polytime disjunction ($\vee$**C**), which will be shown in Table 1. We therefore obtain that KROM[$\vee$] is polynomially translatable to CE-LSNNF. So CE-LSNNF is strictly more succinct than KROM[$\vee$]. The problem whether CE-LSNNF is at least as succinct as renH[$\vee$], K/H[$\vee$] and HORN[$\vee$] remains unknown.

Every CO-LSNNF-formula is an NNF-formula that is CE-logically separable w.r.t. $\perp$. So CO-LSNNF $\leq_s$ CE-LSNNF. But the opposite direction does not hold unless PH collapses.

Proposition 6. CO-LSNNF $\leq_s$ CE-LSNNF *and* CE-LSNNF $\not\leq_s^*$ CO-LSNNF.

CO-LSNNF does not support polytime conditioning which will be shown in Table 1. The following proposition states that CE-LSNNF is the maximal fragment of CO-LSNNF closed under conditioning.

Proposition 7. An NNF-formula $\alpha \in$ CE-LSNNF iff $\alpha \in$ CO-LSNNF and $\alpha|t \in$ CO-LSNNF for every non-trivial term t.

Proof. We prove by induction on α. We here only verify the case where α is an $\wedge$-node $\beta_1 \wedge \cdots \wedge \beta_n$.

($\Rightarrow$): Suppose that $\alpha \in$ CE-LSNNF. Clearly, $\alpha \in$ CO-LSNNF. It remains to verify that $\alpha|t$ is $\mathtt{CO}_{ls}$ for every non-trivial term t. Let c be a non-trivial clause equivalent to $\neg t$. By Definition 4, if $\alpha \not\models c$, then $\alpha|t$ is satisfiable, and hence $\alpha|t \in$ CO-LSNNF. We now assume that $\alpha \models c$. By Definition 7, there is a conjunct β_i of α s.t. $\beta_i \models c$ and β_i is $\mathtt{CE}_{ls}$ w.r.t. c. By the induction hypothesis, we have $\beta_i|t$ is $\mathtt{CO}_{ls}$. By Definition 4, it holds that $\alpha|t$ is $\mathtt{CO}_{ls}$.

($\Leftarrow$): Let $\alpha \in \mathcal{L}$. By assumption, α is CO-logically separable and $\alpha|t$ is also CO-logically separable for every non-trivial term t. Let c' a non-trivial clause and t' a non-trivial term equivalent to $\neg c'$. It remains to verify that α is $\mathtt{CE}_{ls}$ w.r.t. c'. α is an $\wedge$-node $\beta_1 \wedge \cdots \wedge \beta_n$: If $\alpha|t'$ is satisfiable, then $\alpha \not\models c'$. It follows from Definition 7 that α is $\mathtt{CE}_{ls}$ w.r.t. c'. We now assume that $\alpha|t'$ is unsatisfiable. By Definition 4, there is a conjunct β_i of α s.t. $\beta_i|t'$ is unsatisfiable. By the induction hypothesis, we have that β_i is $\mathtt{CE}_{ls}$ w.r.t. c'. Hence, α is $\mathtt{CE}_{ls}$ w.r.t. c'. □

The following proposition provides the upper bound of the CE-LSNNF membership problem.

Proposition 8. *Deciding if an NNF-formula is in CE-LSNNF is in Π_2^p.*

Proof. We first consider the non-membership problem of CE-LSNNF. An NNF-formula α is not a CE-LSNNF-formula iff there exists a non-trivial clause c s.t. $\mathtt{CE}(\alpha, c) = 1$ and $\mathcal{CE}(\alpha, c) = 0$. Since the clausal entailment problem of NNF-formula is coNP-complete, the non-membership problem of CE-LSNNF can be solved by a non-deterministic Turing machine M by invoking an NP oracle that decides if $\mathtt{CE}(\alpha, c) = 1$. Hence, the non-membership problem of CE-LSNNF is in Σ_2^P, and the membership problem is in Π_2^P. □

Comparison to Levesque's Normal Form

In first-order logic, Levesque (1998) defined a class for queries, namely Levesque's normal norm (LNF), such that entailment check is complete, sound and can be tractably solved for a class of knowledge base, namely proper knowledge base, which is equivalent to a possibly infinite consistent set of ground literals. Since the negation $\neg$ is applied to a formula, LNF is not a subset of NNF. To make a clear comparison between LNF and CE-LSNNF, we propose the NNF version of LNF, namely Levesque's negation normal norm (LNNF). We remark that every LNF-formula can be converted into a polysize equivalent one in LNNF, and vice versa.

Definition 8. An NNF-formula α is in LNNF, iff

- α is $\top$, $\bot$, or a literal l; or
- α is an $\wedge$-node $\beta_1 \wedge \cdots \wedge \beta_n$ where
  - for any non-trivial clause c, if $\alpha \models c$, then $\beta_j \models c$ for some $1 \leq j \leq n$; and
  - β_i is in LNNF for every $1 \leq i \leq n$; or
- α is a $\vee$-node $\beta_1 \vee \cdots \vee \beta_n$ where
  - for any non-trivial term t, if $t \models \alpha$, then $t \models \beta_j$ for some $1 \leq j \leq n$; and
  - β_i is in LNNF for every $1 \leq i \leq n$.

We use the formula $\alpha = [((\neg x \vee y) \wedge (\neg y \vee z)) \vee \bot] \wedge (\neg x \vee z)$ to illustrate the distinction between CE-LSNNF and LNNF. The formula α satisfies general CE-logical separability, but is not an LNNF-formula. Let $\beta = (\neg x \vee y) \wedge (\neg y \vee z)$. Each conjunct of β does not entail the clause $\neg x \vee z$, but β does. So β is not in LNNF. It follows that $\beta \vee \bot$ is not in LNNF. Neither does α.

From the above example, we can observe that LNNF incorporates not only a stronger constraint than CE-logical separability for $\wedge$-nodes but also its dual property for $\vee$-nodes. The direct consequences are (1) CE-LSNNF subsumes LNNF, and (2) LNNF supports polytime term implication check. CE-LSNNF is strictly more succinct than LNNF unless PH collapses.

Proposition 9. LNNF $\not\leq_s^*$ CE-LSNNF.

Proof. It can be shown that unless PH collapses, there does not exist a class $\mathcal{L}$ of formulas s.t. every DNF-formula has a polysize equivalent $\mathcal{L}$-formula, and $\mathcal{L}$ supports polytime term implication. This, together with the fact that CE-LSNNF $\leq_s$ DNF and LNNF supports **IM** (Proposition 12), impiles that LNNF $\not\leq_s^*$ CE-LSNNF. $\square$

Model Counting

We now turn to the query model counting that returns the number of models of a propositional formula. A number of key AI tasks can be reduced to the model counting problem, such as probabilistic inference (Chavira and Darwiche 2008) and constraint optimization (Bulatov 2013).

We hereafter develop a unified model counting algorithm.

Definition 9. Let α be an NNF-formula, $\mathbf{Y} \supseteq$ Var(α) and $\mathbf{Y}_0 = \mathbf{Y} \setminus \text{Var}(\alpha)$. The procedure $\mathcal{CT}(\alpha, \mathbf{Y})$ is recursively defined as:

- $\mathcal{CT}(\bot, \mathbf{Y}) = 0$, $\mathcal{CT}(\top, \mathbf{Y}) = 2^{|\mathbf{Y}|}$ and $\mathcal{CT}(l, \mathbf{Y}) = 2^{|\mathbf{Y}|-1}$
- $\mathcal{CT}(\beta_1 \wedge \cdots \wedge \beta_n, \mathbf{Y}) = 2^{|\mathbf{Y}_0|} \cdot \prod_{i=1}^n \mathcal{CT}(\beta_i, \text{Var}(\beta_i))$
- $\mathcal{CT}(\beta_1 \vee \cdots \vee \beta_n, \mathbf{Y}) = \sum_{i=1}^n \mathcal{CT}(\beta_i, \mathbf{Y})$

The above procedure works in a recursive way. The first item is for the base case. The numbers of models of $\top$, $\bot$ and a literal are $2^{|\mathbf{Y}|}$, 0 and $2^{|\mathbf{Y}|-1}$, respectively. In the induction case, the model counting problem of an $\wedge$-node (resp. $\vee$-node) is decomposed to that of its subformula. The number of models of an $\wedge$-node equals to the product of the number of models of its subformula β_i multiplying $2^{|\mathbf{Y}_0|}$. The number of models of a $\vee$-node equals to the sum of the $\mathbf{Y}$-models of its subformula β_i.

Definition 10. Let $\mathcal{L}$ be a language and f a procedure that takes an NNF-formula α and a set of variables $\mathbf{Y}$ s.t. $\mathbf{Y} \supseteq$ Var(α) as inputs, and that returns a natural number. We say

- f is CT-overapproximate for $\mathcal{L}$ iff for every $\mathcal{L}$-formula α, $\texttt{CT}(\alpha, \mathbf{Y}) \leq f(\alpha, \mathbf{Y})$;
- f is CT-underapproximate for $\mathcal{L}$ iff for every $\mathcal{L}$-formula α, $\texttt{CT}(\alpha, \mathbf{Y}) \geq f(\alpha, \mathbf{Y})$.

The procedure $\mathcal{CT}$ is overapproximate for any NNF-formula α with time linear in the size of α.

Theorem 7. *The procedure $\mathcal{CT}$ is* CT*-overapproximate for* NNF *with the time complexity $O(|\alpha|)$.*

However, the procedure $\mathcal{CT}$ may compute the incorrect answer for some NNF-formulas. We illustrate the reasons from the model-theoretic perspective.

We first consider the $\wedge$-node $\alpha = \beta_1 \wedge \cdots \wedge \beta_n$. Let Ω_i be the set of $\mathbf{Y}_i$-models of β_i where $\mathbf{Y}_i = \text{Var}(\beta_i)$. The Cartesian product on n sets of models $\Omega_1, \cdots, \Omega_n$ is defined as: $\Omega_1 \times \cdots \times \Omega_n = \{\omega_1 \cup \cdots \cup \omega_n \mid \omega_i \in \Omega_i \text{ for } 1 \leq i \leq n\}$. The Cartesian product may contain an inconsistent set of literals, that is, a set contains a variable x and its complement $\neg x$. For example, $\{\{x\}\} \times \{\{\neg x, \neg y\}, \{x, y\}\} = \{\{x, \neg x, \neg y\}, \{x, y\}\}$ where $\{x, \neg x, \neg y\}$ is an inconsistent set of literals. The procedure $\mathcal{CT}$ simply returns the product of $|\Omega_1|, \cdots, |\Omega_n|$ and the number $2^{|\mathbf{Y}_0|}$, Therefore, $\mathcal{CT}$ computes an overapproximate result when the Cartesian product $\Omega_1 \times \cdots \times \Omega_n$ contains some inconsistent sets of literals.

We now consider the $\vee$-node $\alpha = \beta_1 \vee \cdots \vee \beta_n$. Let Ω_i be the set of $\mathbf{Y}$-models of β_i The procedure $\mathcal{CT}$ simply returns the sum of $|\Omega_i|$. Therefore, $\mathcal{CT}$ produces an overapproximate result since a $\mathbf{Y}$-interpretation may satisfy more than one disjunct of α.

In the following, we identify the conditions under which the procedure $\mathcal{CT}$ provides the exact counting of models. We say two formulas α and β agree on common variables, if for every variable $x \in \text{Var}(\alpha) \cap \text{Var}(\beta)$, we have $\alpha \models x$ and $\beta \models x$; or $\alpha \models \neg x$ and $\beta \models \neg x$.

Definition 11. An NNF-formula is CT-logically separable ($\texttt{CT}_{ls}$) iff one of the following conditions hold

- α is a literal, $\top$ or $\bot$;
- α is an $\wedge$-node $\beta_1 \wedge \cdots \wedge \beta_n$ s.t.
  - if α is unsatisfiable, then β_i is unsatisfiable and $\texttt{CT}_{ls}$ for some i;
  - if α is satisfiable, then every β_i is $\texttt{CT}_{ls}$ and every two distinct β_i and β_j agree on common variables.
- α is an $\vee$-node $\beta_1 \vee \cdots \vee \beta_n$ s.t. $\beta_i \wedge \beta_j \models \bot$ for $i \neq j$ and every β_i is $\texttt{CT}_{ls}$.

With the above constraints, no inconsistent set of literals exists in the Cartesian product on sets of models of conjuncts of $\wedge$-nodes. Neither common models in some distinct disjuncts of $\vee$-nodes does. We use CT-LSNNF to denote the subset of NNF in which any formula satisfies CT-logical separability property. CT-LSNNF precisely capture the class of formulas that allow $\mathcal{CT}$ to produce a correct result.

Theorem 8. *A language $\mathcal{L}$ is* CT-LSNNF *iff the procedure $\mathcal{CT}$ is* CT*-overapproximate and* CT*-underapproximate for $\mathcal{L}$.*

Proof. We prove by induction on α. We here only verify the case where α is an $\wedge$-node $\beta_1 \wedge \cdots \wedge \beta_n$.

($\Rightarrow$): Let α be a CT-LSNNF-formula and $\mathbf{Y} \supseteq \text{Var}(\alpha)$. We prove that $\mathcal{CT}(\alpha, \mathbf{Y}) = |\text{Mod}_{\mathbf{Y}}(\alpha)| = \texttt{CT}(\alpha, \mathbf{Y})$. The induction step for $\wedge$-node: Let $\beta_0 = \top$, $\mathbf{Y}_0 = \mathbf{Y} \setminus \text{Var}(\alpha)$ and $\mathbf{Y}_i = \text{Var}(\beta_i)$ for $1 \leq i \leq n$. In the case where α is satisfiable. By CT-logically separability property, we get that $\text{Mod}_{\mathbf{Y}_0}(\beta_0) \times \text{Mod}_{\mathbf{Y}_1}(\beta_1) \times \cdots \times \text{Mod}_{\mathbf{Y}_n}(\beta_n)$ does not contain an inconsistent set of literals. Hence, the union of each $\mathbf{Y}_i$-model of β_i is a $\mathbf{Y}$-model of α. By the inductive hypothesis, we get that $\mathcal{CT}(\beta_i, \mathbf{Y}_i) = |\text{Mod}_{\mathbf{Y}_i}(\beta_i)|$ for $1 \leq i \leq n$. So $|\text{Mod}_{\mathbf{Y}}(\alpha)| = \prod_{i=0}^{n} |\text{Mod}_{\mathbf{Y}_i}(\beta_i)| = 2^{|\mathbf{Y}_0|} \cdot \prod_{i=1}^{n} \mathcal{CT}(\beta_i, \mathbf{Y}_i) = \mathcal{CT}(\alpha, \mathbf{Y})$. In the case where α is unsatisfiable. By $\texttt{CT}_{ls}$ property and the inductive hypothesis, it is easy to verify that $\mathcal{CT}(\alpha, \mathbf{Y}) = |\text{Mod}_{\mathbf{Y}}(\alpha)| = 0$.

($\Leftarrow$): Suppose that the language $\mathcal{L}$ is such that for every $\mathcal{L}$-formula α and every $\mathbf{Y} \supseteq \text{Var}(\alpha)$, $\texttt{CT}(\alpha, \mathbf{Y}) = \mathcal{CT}(\alpha, \mathbf{Y})$. We prove that every $\mathcal{L}$-formula α is CT-logically separable. We w.l.o.g. assume that $\mathbf{Y} = \text{Var}(\alpha)$. Let $\mathbf{Y}_i = \text{Var}(\beta_i)$ for $1 \leq i \leq n$. In the case α is satisfiable. By the assumption $\texttt{CT}(\alpha, \mathbf{Y}) = \mathcal{CT}(\alpha, \mathbf{Y})$, the following hold:

1. $\texttt{CT}(\beta_i, \mathbf{Y}_i) = \mathcal{CT}(\beta_i, \mathbf{Y}_i)$ for every $1 \leq i \leq n$;
2. $\text{Mod}_{\mathbf{Y}_1}(\beta_1) \times \cdots \times \text{Mod}_{\mathbf{Y}_n}(\beta_n)$ does not contain an inconsistent set of literals.

By Item 1 and the induction hypothesis, we get that each β_i is $\texttt{CT}_{ls}$. By Item 2, we get that every two distinct formulas β_i and β_j agree on common variables. It follows from CT-logically separability property that $\alpha \in$ CT-LSNNF.

In the case where α is unsatisfiable. By the assumption, we get that $\mathcal{CT}(\alpha, \mathbf{Y}) = \texttt{CT}(\alpha, \mathbf{Y}) = 0$. By procedure $\mathcal{CT}$, $\mathcal{CT}(\beta_i, \mathbf{Y}_i) = 0$ for some i. Since the $\mathcal{CT}$ procedure is CT-overapproximate, β_i is unsatisfiable. So $\mathcal{CT}(\beta_i, \mathbf{Y}_i) = \texttt{CT}(\beta_i, \mathbf{Y}_i)$. By the induction hypothesis, β_i is $\texttt{CT}_{ls}$. Hence, $\alpha \in$ CT-LSNNF. ☐

Darwiche (2001b) proposed a model counting algorithm that is correct for sd-DNNF, that is, the subset of d-DNNF with the smoothness property requiring that $\text{Var}(\beta_i) = \text{Var}(\beta_j)$ for each $\vee$-node $\beta_1 \vee \cdots \vee \beta_n$ of an NNF-formula. By comparison, the $\mathcal{CT}$ algorithm has wider scope of application and is more efficient than (Darwiche 2001b)'s algorithm since CT-LSNNF is a strict superset of sd-DNNF and transforming a CT-LSNNF-formula into an equivalent sd-DNNF-formula may cause a $O(|\mathbf{X}|)$ blowup in size where $|\mathbf{X}|$ is the number of variables.

We hereafter prove that d-DNNF and CT-LSNNF are polynomially equivalent.

Proposition 10. d-DNNF $\sim_p$ CT-LSNNF.

Proof. It is easily verified that CT-LSNNF $\leq_p$ d-DNNF. We here only prove that d-DNNF $\leq_p$ CT-LSNNF. We can transform every CT-LSNNF-formula to an equivalent d-DNNF-formula in a bottom-up manner. During the transformation, we only convert every $\wedge$-subnode $\alpha = \bigcup_{i=1}^{n} \beta_i$ as follows. We first determine the satisfiability of α, which can be solved in polytime. If α is unsatisfiable, then $\perp$ is a d-DNNF-formula equivalent to α. Otherwise, it follows from the CT-logically separability property that each β_i is in CT-LSNNF. Suppose that β'_i is the transformed d-DNNF-formula equivalent to β_i with polysize in $|\beta_i|$. Let $\mathbf{L} = \bigcup_{i=1}^{n}[\text{Lit}(\beta'_i) \cap (\bigcup_{j=1}^{i-1} \text{Lit}(\beta'_j))]$ and $\alpha' = \beta'_1|\mathbf{L} \wedge \cdots \wedge \beta'_n|\mathbf{L} \wedge \mathbf{L}$. Since d-DNNF is closed under conditioning (cf. Proposition 5.1 in (Darwiche and Marquis 2002)), $\beta'_i|\mathbf{L}$ is also d-DNNF. It is easily verified that $\alpha' \equiv \alpha$ and α' satisfies determinism and decomposability. ☐

To the best of our knowledge, the maximal fragment of NNF supporting polytime model counting, which is discovered in the existing literature, is d-DNNF (Darwiche 2001b).

In the following, we prove a dual language, namely c-$\bar{\text{D}}$NNF, to d-DNNF that is not a subset of CT-LSNNF but supports polytime model counting. The $\vee$-decomposability property requires the sets of variables of the children of each $\vee$-node to be pairwise disjoint. The covering property requires the children of each $\wedge$-node in a formula to be pairwise logically tautology. *Cover dual decomposability NNF (*c-$\bar{\text{D}}$NNF*)* is the subset of NNF satisfying $\vee$-decomposability and covering properties.

We now design a polytime model counting procedure $\mathcal{CT}'$ for c-$\bar{\text{D}}$NNF. Every c-$\bar{\text{D}}$NNF-formula α can be transformed into a d-DNNF-formula $\bar{\alpha}$ in polytime s.t. $\alpha \equiv \neg\bar{\alpha}$ via replacing any occurrence of Boolean constants $\top$ (resp. $\perp$) by $\perp$ (resp. $\top$), of literals l by its complement $\bar{l}$ and of connectives $\wedge$ (resp. $\vee$) by $\vee$ (resp. $\wedge$). For convenience, we call the d-DNNF-formula $\bar{\alpha}$, generated from α by the above process, the complement of α. The procedure $\mathcal{CT}'$ for c-$\bar{\text{D}}$NNF is defined as $\mathcal{CT}'(\alpha, \mathbf{Y}) = 2^{|\mathbf{Y}|} - \mathcal{CT}(\bar{\alpha}, \mathbf{Y})$. Since the formula $\bar{\alpha}$ is d-DNNF, the procedure $\mathcal{CT}$ produces a exact number of models of $\bar{\alpha}$. Hence, $2^{|\mathbf{Y}|} - \mathcal{CT}(\bar{\alpha}, \mathbf{Y})$ is the correct answer for the models of α.

We illustrate the fact that c-$\bar{\text{D}}$NNF is not CT-logical separable with the formula $\alpha = (x \wedge \neg x) \vee y$. The subformula $x \wedge \neg x$ of α is unsatisfiable, but neither x nor $\neg x$ is. It violates the requirement of unsatisfiable $\wedge$-nodes in CT-LSNNF, and hence is not in CT-LSNNF. However, this formula α satisfies $\vee$-decomposability and covering.

Finally, we prove that every language $\mathcal{L}$ that is polynomially translatable into CT-LSNNF also satisfies **CT**. But the opposite direction remains open.

Theorem 9. *If a language $\mathcal{L}$ is polynomially translatable to* CT-LSNNF*, then $\mathcal{L}$ satisfies* **CT**.

We close this section by providing the upper bound of the CT-LSNNF membership problem.

Proposition 11. *Deciding if an* NNF*-formula is in* CT-LSNNF *is in Δ_2^P.*

Proof. It is easy to design a deterministic algorithm for the CT-LSNNF membership problem via invoking NP oracle according to the definition of CT-LSNNF-formulas (cf. Definition 11). Identify whether an NNF-formula α is in CT-LSNNF requires checking the satisfiability of some subformulas of α and the conjunction of some subformulas, and checking the entailment of some subformulas and some literals. NP oracles are used to solve the above two checking and are invoked at most $O(n^4)$ times where n is the size of α. ☐

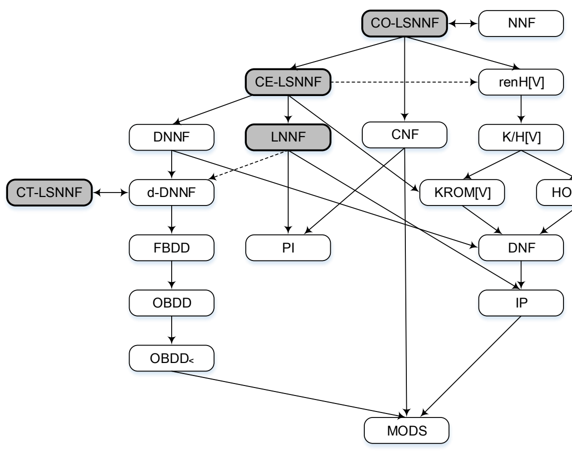

Figure 1: Succinctness of various normal forms. An edge $\mathcal{L}_1 \rightarrow \mathcal{L}_2$ indicates $\mathcal{L}_1 <_s \mathcal{L}_2$ while an edge $\mathcal{L}_1 \leftrightarrow \mathcal{L}_2$ indicates $\mathcal{L}_1 \sim_s \mathcal{L}_2$. A dotted right arrow from $\mathcal{L}_1$ to $\mathcal{L}_2$ means $\mathcal{L}_2 \not\leq_s \mathcal{L}_1$ but whether $\mathcal{L}_1 \leq_s \mathcal{L}_2$ is unknown.

$\mathcal{L}$	**CO**	**VA**	**CE**	**IM**	**EQ**	**SE**	**CT**	**ME**
CO-LSNNF	√	○	○	○	○	○	○	○
CE-LSNNF	√	○	√	○	○	○	○	√
LNNF	√	√	√	√	○	○	○	√
CT-LSNNF	√	√	√	√	?	○	√	√
$\mathcal{L}$	**CD**	**FO**	**SFO**	**∧BC**	**∧C**	**∨BC**	**∨C**	**¬C**
CO-LSNNF	○	○	√	○	○	√	√	○
CE-LSNNF	√	√	√	○	○	√	√	○
LNNF	√	?	?	○	○	○	○	√
CT-LSNNF	√	○	○	○	○	○	○	?

Table 1: Queries and transformations for the logical separability based languages. An occurrence of $\surd$ indicates that $\mathcal{L}$ supports the query (or transformation) property; $\circ$ means that it is not the case unless $\mathsf{P} = \mathsf{NP}$; and ? means that whether $\mathcal{L}$ supports this property is unknown.

Succinctness, Queries and Transformations

As pointed out in (Darwiche and Marquis 2002), succinctness, queries and transformations are three key dimensions to consider when choosing an appropriate target language for application-specific problems. In this section, we discuss the relative succinctness of logical separability based languages (CO-LSNNF, CE-LSNNF, LNNF and CT-LSNNF) compared to other languages considered in (Darwiche and Marquis 2002; Fargier and Marquis 2014) in Figure 1. We also investigate the supported queries and transformations of logical separability based languages in Table 1.

Proposition 12. The results in Figure 1 and Table 1 hold.

We can make several observations from Figure 1 and Table 1 as follows. (1) There are two succinctness orderings of logical separability based languages: CO-LSNNF $<_s$ CE-LSNNF $<_s$ CT-LSNNF and CO-LSNNF $<_s$ CE-LSNNF $<_s$ LNNF. CT-LSNNF $\not\leq_s$ LNNF but whether LNNF $\leq_s$ CT-LSNNF remains open. (2) Adding CO-logical separability to NNF gains the polytime satisfiability check property without lowering down its succinctness hierarchy. But CO-LSNNF loses the transformation properties: **CD**, **∧BC**, **∧C** and **¬C** which hold in NNF. It is worthy noting that CO-LSNNF is at least as succinct as any language that offers polytime satisfiability check. (3) Besides KROM[∨] and subsets of DNNF, CE-LSNNF is another fragment of NNF supporting **CE**, **FO** and **∨C**. Furthermore, CE-LSNNF is strictly more succinct than any languages supporting **CE** except renH[∨], K/H[∨] and HORN[∨]. (4) Although LNNF offers **VA** and **IM** compared to CE-LSNNF, but the former is not closed under bounded disjunction and is less succinct than the latter. (5) CT-LSNNF is equally succinct to d-DNNF and hence supports the same query and transformation properties as d-DNNF. (6) None of logical separability based languages satisfies bounded conjunction.

Conclusion and Future Work

Inspired by Levesque, we introduce three new logical separability based properties: CO-logical separability, CE-logical separability and CT-logical separability for query tasks CO, CE and CT respectively, establishing the connection between knowledge compilation and logical separability. We then identify three novel fragments: CO-LSNNF, CE-LSNNF and CT-LSNNF, which precisely capture the classes of formulas that permit polytime procedures $\mathcal{CO}$, $\mathcal{CE}$ and $\mathcal{CT}$ always produce a correct answer of the corresponding tasks respectively. We also extend the knowledge compilation map by investigating succinctness and tractability of the four logical separability based languages (CO-LSNNF, CE-LSNNF, LNNF and CT-LSNNF). We show that CO-LSNNF is the most succinct language permitting **CO** and particularly any propositional formula has a polysize equivalent CO-LSNNF formula. CE-LSNNF is as powerful as DNNF since they supports the same set of queries and transformations, yet strictly more succinct. CT-LSNNF is the same as succinct as d-DNNF and offers the same tractability as d-DNNF.

There are several directions for future works. Firstly, the remaining open questions include: the unknown succinctness, supported queries and transformations which shown in Figure 1 and Table 1 as well as the lower bound of the membership problem of CE-LSNNF and CT-LSNNF. Secondly, we would like to develop a compilation method for CE-LSNNF, leading to build up a more efficient representation for two important AI problems: planning and model-based diagnosis. This is because that CE-LSNNF is the most succinct normal form, shown in Figure 1, which supports the four basic queries and transformations (clausal entailment, model enumeration, conditioning and forgetting) that are essential to planning and model-based diagnosis. Finally, it is interesting to further apply the notion of logical separability to more query and transformation tasks.

Acknowledgments

This paper was supported by the National Natural Science Foundation of China (Nos. 62077028, 61877029, 61906216, 62176103 and 61603152), the Guangdong Basic and Applied Basic Research Foundation (Nos. 2021A1515011873, 2021B0101420003, 2021B1515120048, 2020A1515010642, 2020B0909030005, 2020B1212030003, 2020ZDZX3013, 2019A050510024, 2019B1515120010 and 2018A030310633), the Specially Creative University Project of Guangdong (Nos. 2019KTSCX010 and 2018KTSCX016), the Fundamental Research Funds for the Central Universities (Nos. 11621418 and 21617349), the Science and Technology Project of Guangzhou (Nos. 202102080307, 202102021173 and 201902010041), and the Project of Guangxi Key Laboratory of Trusted Software (No. kx201604).

References

Berre, D. L.; Marquis, P.; Mengel, S.; and Wallon, R. 2018. Pseudo-Boolean Constraints from a Knowledge Representation Perspective. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-2018)*, 1891–1897.

Bulatov, A. A. 2013. The Complexity of the Counting Constraint Satisfaction Problem. *Journal of the ACM*, 60(5): 1–41.

Chavira, M.; and Darwiche, A. 2008. On Probabilistic Inference by Weighted Model Counting. *Artificial Intelligence*, 172(6-7): 772–799.

Darwiche, A. 2001a. Decomposable Negation Normal Form. *Journal of the ACM*, 48(4): 608–647.

Darwiche, A. 2001b. On the Tractable Counting of Theory Models and its Application to Truth Maintenance and Belief Revision. *Journal of Applied Non-Classical Logics*, 11(1-2): 11–34.

Darwiche, A. 2011. SDD: A New Canonical Representation of Propositional Knowledge Bases. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI-2011)*, 819–826.

Darwiche, A.; and Hirth, A. 2020. On the Reasons Behind Decisions. In *Proceedings of the Twenty-Fourth European Conference on Artificial Intelligence (ECAI-2020)*, 712–720.

Darwiche, A.; and Marquis, P. 2002. A Knowledge Compilation Map. *Journal of Artificial Intelligence Research*, 17(1): 229–264.

Davis, M.; and Putnam, H. 1960. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7(3): 201–215.

Fargier, H.; and Marquis, P. 2009. Knowledge Compilation Properties of Trees-of-BDDs, Revisited. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-2009)*, 772–777.

Fargier, H.; and Marquis, P. 2014. Disjunctive Closures for Knowledge Compilation. *Artificial Intelligence*, 216: 129–162.

Horn, A. 1951. On Sentences Which are True of Direct Unions of Algebras. *Journal of Symbolic Logic*, 16(3): 14–21.

Huang, J. 2006. Combining Knowledge Compilation and Search for Conformant Probabilistic Planning. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS-2006)*, 253–262.

Huang, J.; and Darwiche, A. 2005. On Compiling System Models for Faster and More Scalable Diagnosis. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-2005)*, 300–306.

Krom, M. R. 1970. The Decision Problem for Formulas in Prenex Conjunctive Normal Form with Binary Disjunctions. *Journal of Symbolic Logic*, 35(2): 210–216.

Levesque, H. J. 1998. A Completeness Result for Reasoning with Incomplete First-Order Knowledge Bases. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR-1998)*, 14–23.

Marquis, P. 2000. Consequence Finding Algorithms. In *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, 41–145. Springer.

Martires, P. Z. D.; Dries, A.; and De Raedt, L. 2019. Exact and Approximate Weighted Model Integration with Probability Density Functions using Knowledge Compilation. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-2019)*, 7825–7833.

Mateescu, R.; Dechter, R.; and Marinescu, R. 2008. AND/OR Multi-Valued Decision Diagrams (AOMDDs) for Graphical Models. *Journal of Artificial Intelligence Research*, 33: 465–519.

Palacios, H.; Bonet, B.; Darwiche, A.; and Geffner, H. 2005. Pruning Conformant Plans by Counting Models on Compiled d-DNNF Representations. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS-2005)*, 141–150.

Pipatsrisawat, K.; and Darwiche, A. 2008. New Compilation Languages Based on Structured Decomposability. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*, 517–522.

Selman, B.; and Kautz, H. 1996. Knowledge Compilation and Theory Approximation. *Journal of the ACM*, 43(2): 193–224.

Shih, A.; Choi, A.; and Darwiche, A. 2019. Compiling Bayesian Network Classifiers into Decision Graphs. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-2019)*, 7966–7974.

Shih, A.; Darwiche, A.; and Choi, A. 2019. Verifying Binarized Neural Networks by Angluin-Style Learning. In *Proceedings of the Twenty-Second International Conference on Theory and Applications of Satisfiability (SAT-2019)*, volume 11628, 354–370.

Čepek, O.; and Chromý, M. 2020. Properties of Switch-List Representations of Boolean Functions. *Journal of Artificial Intelligence Research*, 69: 501–529.

Wachter, M.; and Haenni, R. 2006. Propositional DAGs: A New Graph-Based Language for Representing Boolean Functions. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR-2006)*, 277–285.