On the Complexity of Enumerating Prime Implicants from Decision-DNNF Circuits

Alexis de Colnet[1] and **Pierre Marquis**[1,2]

[1]Univ. Artois, CNRS, Centre de Recherche en Informatique de Lens (CRIL), F-62300 Lens, France
[2]Institut Universitaire de France
{decolnet, marquis}@cril.fr

Abstract

We consider the problem Enum·*IP* of enumerating prime implicants of Boolean functions represented by decision decomposable negation normal form (dec-DNNF) circuits. We study Enum·*IP* from dec-DNNF within the framework of enumeration complexity and prove that it is in OutputP, the class of output polynomial enumeration problems, and more precisely in IncP, the class of polynomial incremental time enumeration problems. We then focus on two closely related, but seemingly harder, enumeration problems where further restrictions are put on the prime implicants to be generated. In the first problem, one is only interested in prime implicants representing subset-minimal abductive explanations, a notion much investigated in AI for more than thirty years. In the second problem, the target is prime implicants representing sufficient reasons, a recent yet important notion in the emerging field of eXplainable AI, since they aim to explain predictions achieved by machine learning classifiers. We provide evidence showing that enumerating specific prime implicants corresponding to subset-minimal abductive explanations or to sufficient reasons is not in OutputP.

1 Introduction

Prime implicants are a key concept when dealing with Boolean functions since the notion has been introduced seven decades ago [Quine, 1952]. Within AI, prime implicants (or the dual concept of prime implicates) have been considered for modeling and solving a number of problems, including compiling knowledge [Reiter and de Kleer, 1987] and generating explanations of various kinds. This is the case in logic-based abductive reasoning (see e.g., [Selman and Levesque, 1990; Eiter and Gottlob, 1995]), a form of inference required in many applications when the available knowledge base is incomplete (e.g., in medicine) and because of such an incompleteness, it cannot alone explain the observations that are made about the state of the world. Abduction gave rise to much research in AI for the past thirty years, especially because it is closely connected to other reasoning settings, including truth maintenance [de Kleer, 1986], assumption-based reasoning and closed-world reasoning (see e.g., [Marquis, 2000] for a survey). Formally, the explanations one looks for are terms over a preset alphabet (composed of the so-called abducible variables, e.g., representing diseases) such that the manifestations that are reported (e.g., some symptoms) are logical consequences of the background knowledge when completed by such a term. In order to avoid trivial explanations, one also asks those terms to be consistent with the knowledge base. Explanations that are the less demanding ones from a logical standpoint (i.e., subset-minimal ones) can be characterized as specific prime implicants. More recently, deriving explanations justifying why certain predictions have been made has appeared as essential for ensuring trustworthy Machine Learning (ML) technologies [Miller, 2019; Molnar, 2019]. In the research area of eXplainable AI (XAI), recent work has shown how ML classifiers of various types (including black boxes) can be associated with Boolean circuits (alias transparent or "white" boxes), exhibiting the same input-output behaviours [Narodytska *et al.*, 2018; Shih *et al.*, 2018a; Shih *et al.*, 2019]. Thanks to such mappings, XAI queries about classifiers can be delegated to the corresponding circuits. The notion of sufficient reason for an instance given a Boolean function f modeling a binary classifier has been introduced in [Darwiche and Hirth, 2020]. Given an instance a (a simply is an assignment, i.e., a vector of truth values given to each of the n variables) such that $f(a) = 1$ (resp. $f(a) = 0$), a sufficient reason for a is a subset-minimal partial assignment a' which is coherent with a (i.e., a and a' give the same values to the variables that are assigned in a') and which satisfies the property that for every extension a'' of a' we have $f(a'') = 1$ (resp. $f(a'') = 0$). The features assigned in a' (and the way they are assigned) can be viewed as explaining why a has been classified by f as a positive (or as a negative) instance.

Whatever the way prime implicants are used, generating them is in general a computationally demanding task, for at least two reasons. On the one hand, deriving a single prime implicant of a Boolean function represented by a propositional formula (or circuit) is NP-hard since such a formula is satisfiable when it has a prime implicant, and it is valid precisely when this prime implicant is the empty term. On the other hand, a source of complexity is the number of prime implicants that may prevent from computing them all. In-

deed, it is well-known that the number of prime implicants of a Boolean function can be exponential in the number of variables of the function, and, for many representations of the function, also exponential in the size of the representation (just consider the parity function as a matter of example). In more detail, the number of prime implicants of a Boolean function can be larger than the number of assignments satisfying the function [Dunham and Fridshal, 1959]; there also exist families of Boolean functions over n variables having $\Omega(\frac{3^n}{n})$ prime implicants [Chandra and Markowsky, 1978].

In this paper, we focus on the issue of *enumerating prime implicants of a Boolean function represented by a decision decomposable negation normal form circuit (alias a Decision-DNNF circuit – dec-DNNF circuit for short)*. The question is to determine whether such prime implicants *can be enumerated "efficiently"*, which is obviously not the case when the circuit considered is unconstrained (as explained above, in such a case, computing a single prime implicant is already hard). This question is important for all the problems listed previously, when prime implicants represent explanations: since they are typically too numerous to be computed as a whole, it makes sense to derive them in an incremental way, with some performance guarantees in the generation; this lets the user who asked for an explanation deciding what to do after each derivation, namely to stop the enumeration process since he/she is satisfied by the explanation that has been provided, or alternatively to ask for a further explanation.

The dec-DNNF language [Oztok and Darwiche, 2014; Darwiche, 2001] and its subsets FBDD (free binary decision diagrams) [Gergov and Meinel, 1994], OBDD (ordered binary decision diagrams) [Bryant, 1986] and even DT (the set of all binary decision trees over Boolean variables, see e.g., [Wegener, 2000, Chapter 2]) appear at first sight as good candidates for representing the function in the perspective of enumerating "efficiently" its prime implicants. Indeed, they are known as tractable representation languages (they support in polynomial time many queries and transformations from the so-called knowledge compilation map [Darwiche and Marquis, 2002; Koriche *et al.*, 2013]).

The main contribution of the paper is as follows. We give a polynomial incremental time algorithm for enumerating the prime implicants of a Boolean function f represented by a dec-DNNF circuit Σ. Given Σ and a positive integer k, this algorithm returns k prime implicants of Σ in $O(poly(k + |\Sigma|))$ time, or returns all prime implicants of Σ if there are fewer than k. This shows that enumerating prime implicants from dec-DNNF is in the enumeration complexity class IncP [Strozecki, 2019]. We also provide evidence showing that enumerating specific prime implicants corresponding to subset-minimal abductive explanations or to sufficient reasons is not in OutputP: on the one hand, computing a single subset-minimal abductive explanation from an OBDD circuit or a decision tree is NP-hard; on the other hand, the existence of an output polynomial time algorithm for enumerating sufficient reasons given an OBDD circuit or a decision tree would lead to an output polynomial time algorithm for enumerating the minimal transversals of a hypergraph, thus answering a long-standing question related to monotone dualization [Eiter *et al.*, 2008].

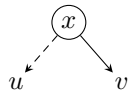
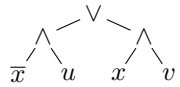
The rest of the paper is organized as follows. We start with some preliminaries (Section 2) where the language of dec-DNNF circuits and the framework used to study enumeration problems are presented. We formally define the problem Enum·*IP* of enumerating prime implicants. Then in Section 3 we show that generating the set of *all* prime implicants from a dec-DNNF circuit is feasible in output polynomial time. From there, we show in Section 4 that Enum·*IP* from dec-DNNF is in fact in IncP and point out a polynomial incremental time enumeration algorithm. Finally, in Section 5 we focus on subset-minimal abductive explanations and sufficient reasons and show that for each of the two cases, the enumeration issue is seemingly harder than in the case when all prime implicants are considered. A full-proof version of the paper is available at www.cril.univ-artois.fr/expekctation/papers.html.

2 Preliminaries

A Boolean function over n variables $x_1, \ldots, x_n$ is a mapping f from $\{0,1\}^n$ to $\{0,1\}$. The set of variables of f is denoted by $var(f)$. The assignments to $var(f)$ mapped to 1 by f are called *satisfying assignments* of f. A literal upon variable x is either x or its negation $\overline{x}$ and a *term* is a conjunction of literals. We often omit the conjunction symbols when writing terms, for instance we may shorten $a \wedge \overline{c}$ into $a\,\overline{c}$. We define the *empty* term $t_\emptyset$ as the term over zero literal. The empty term verifies $t \wedge t_\emptyset = t$ for every term t. Given $\ell \in \{x, \overline{x}\}$, we denote by $f|\ell$ the Boolean function over $var(f) \setminus \{x\}$ whose satisfying assignments coincide with that of $f \wedge \ell$. We use the usual symbols $\wedge$, $\vee$, $\neg$, $\models$ to denote conjunction, disjunction, negation, and entailment. Given a set S of terms, $\max(S, \models)$ denotes the subset of terms of S that do not entail another term in S. An *implicant* of a Boolean function f is a term t whose satisfying assignments also satisfy f, i.e., $t \models f$. An implicant t is *prime* when no term $t - \ell$ obtained by removing a literal ℓ from t is an implicant of f.

2.1 Compilation Languages

Compilation languages are often seen as classes of circuits. Let PS be a countable set of propositional variables. A circuit in *negation normal form* (NNF) is a directed acyclic graph (DAG) whose leaves are labelled with 0 (*false*), 1 (*true*), or a literal built upon $x \in PS$, and whose internal nodes are labelled with $\wedge$ or $\vee$ connectives; we call them $\wedge$-nodes and $\vee$-nodes. An NNF circuit computes a Boolean function over the variables appearing in it. For v a node of an NNF circuit Σ, $var(v)$ denotes the set of variables labelling leaves under v in Σ and Σ_v denotes the subcircuit of Σ rooted at v. The language of *decomposable* NNF circuits (DNNF) contains the NNF circuits where $\wedge$-nodes are decomposable, that is, the children $v_1, \ldots, v_m$ of every $\wedge$-node v are such that $var(v_i) \cap var(v_j) = \emptyset$ for all $i \neq j$. The language of *deterministic, decomposable* NNF circuits (d-DNNF) contains the DNNF circuits Σ where $\vee$-nodes are deterministic, that is, the children $v_1, \ldots, v_m$ of every $\vee$-node v are such that $\Sigma_{v_i} \wedge \Sigma_{v_j}$ is inconsistent for all $i \neq j$. Finally,

the language of *Decision*-DNNF circuits (dec-DNNF) is that of circuits whose leaves are labelled with 0, 1, or a literal built upon $x \in PS$, and whose internal nodes are decision nodes and $\wedge$-nodes. Whenever n is a decision node labelled by variable x in a dec-DNNF circuit Σ, the circuit Σ_n given by  is viewed as a compact representation of the d-DNNF circuit  (see Definition 2.6 and Figure 2 in [Darwiche and Marquis, 2002]).

Thus, a decision node n is labelled by a variable and has two children: the 0-child (node u on the previous picture) and the 1-child (node v on the previous picture). If n is labelled by x and Σ_u (resp. Σ_v) represents the function f_0 (resp. f_1), then Σ_n represents the function $(\overline{x} \wedge f_0) \vee (x \wedge f_1)$. For instance, Figure 1a gives a dec-DNNF circuit whose deepest decision node computes $(\overline{s} \wedge 1) \vee (s \wedge p)$. It is worth mentioning that all Boolean functions on finitely many variables can be represented in dec-DNNF, or indeed in any of its subsets like FBDD, OBDD, and DT.

Let Σ be a dec-DNNF circuit. The size of Σ, denoted by $|\Sigma|$ is its number of edges. From a dec-DNNF circuit Σ, one can easily derive in polynomial time a dec-DNNF circuit equivalent to Σ where every $\wedge$-node has exactly two children. Since it is computationally harmless, for the sake of simplicity, our enumeration algorithms suppose that the dec-DNNF circuits satisfy this condition, so that their size is at most twice their number of nodes. In the same vein, we suppose that our dec-DNNF circuits have been *reduced*, i.e., every node v in Σ such that Σ_v computes the 0 function reduces to a leaf labelled by 0. Testing the satisfiability of a dec-DNNF circuit is feasible in linear time [Darwiche and Marquis, 2002], so reducing a dec-DNNF circuit also is a polynomial-time operation.

2.2 Enumeration Complexity

We now recall some enumeration complexity classes as described in [Strozecki, 2019]. Let V be an alphabet and let A be a binary predicate in $V^* \times V^*$. Given an instance $x \in V^*$ (the input), $A(x)$ (the set of solutions) denotes the set of all $y \in V^*$ such that $A(x, y)$. The enumeration problem Enum·A is the function mapping x to $A(x)$. Enum·A is in the class EnumP if for every $y \in A(x)$, $|y|$ is polynomial in $|x|$, and if deciding whether y is in $A(x)$ is in P. EnumP does not capture the complexity of *computing* the set of solutions $A(x)$, it serves more as a counterpart of NP for enumeration problems.

The model used for the enumeration of solutions is the random access machine (RAM) model. See [Strozecki, 2019] for details on why RAM have been chosen for this task. A RAM solves Enum·A if, for all x, it returns a sequence $y_1, \ldots, y_m$ of pairwise distinct elements such that $\{y_1, \ldots, y_m\} = A(x)$. Enum·$A$ is in OutputP if there is a RAM solving Enum·A in time $O(poly(|x| + |A(x)|))$ on every input x. OutputP is a relevant enumeration class when the whole set of solutions is explicitly asked for. For instance, the dualization of a monotone CNF formula ϕ is the task of generating a DNF formula equivalent to ϕ. Because of the monotony condition on ϕ, the terms used in any smallest DNF formula equivalent to ϕ are precisely its prime implicants. Thus, the dualization problem boils down to enumerating *all* the prime implicants of ϕ.

For other applications, computing only a fixed number of solutions may be enough. A RAM solves Enum·A in incremental time $f(t)g(n)$ if on every x, it runs in time $O(f(t)g(|x|))$ and returns a sequence $y_1, \ldots, y_t$ of t pairwise distinct elements of $A(x)$ when $t \leq |A(x)|$, and the whole set $A(x)$ when $t > |A(x)|$. Enum·A is in IncP if there is a RAM that solves A in incremental time $O(t^a n^b)$ for some constants a and b. IncP has a characterization that uses the function problem AnotherSol·A which, given x and $S \subseteq A(x)$, returns $y \in A(x) \setminus S$ when $S \neq A(x)$, and *false* otherwise.

Proposition 1 ([Strozecki, 2019])**.** *A problem Enum·A in* EnumP *is in* IncP *if and only if AnotherSol·A is in* FP.

Note that OutputP is thought to be distinct from IncP [Strozecki, 2019].

3 Enum·IP from Dec-DNNF is in OutputP

Let us first consider the problem of enumerating the prime implicants of a Boolean function f given as a dec-DNNF circuit Σ, for short the prime implicants of Σ. Let *IP*(Σ, t) be the binary predicate representing the relation that t is a prime implicant of Σ. Then *IP*(Σ) denotes the set of prime implicants of Σ. We extend the notation *IP*$(\cdot)$ to any Boolean function f. To be able to speak of prime implicants enumeration from circuits other than dec-DNNF ones we write "Enum.*IP* from L" with L the language Σ belongs to.

We start with a couple of easy results. First of all, since there is a linear-time procedure to verify that a term is an implicant of a dec-DNNF circuit, there is a polynomial-time algorithm to decide whether a given a term is a prime implicant of a dec-DNNF circuits, thus:

Proposition 2. *Enum·IP from dec-DNNF is in* EnumP.

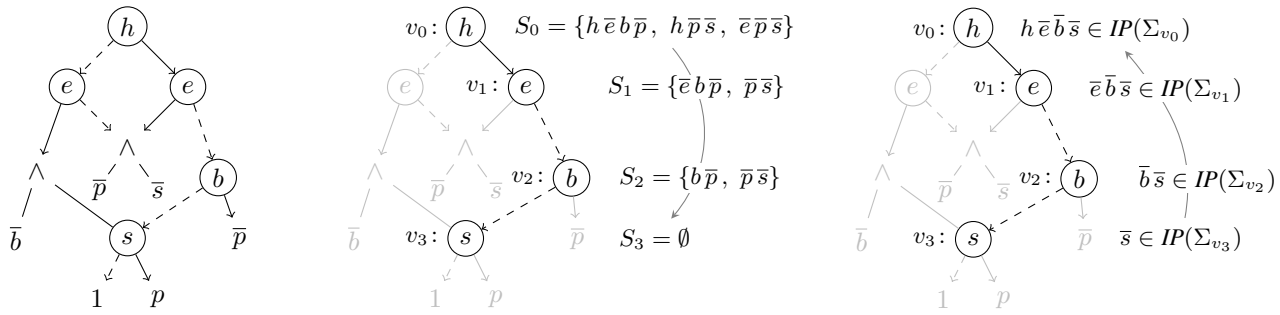
In addition, it is known that Enum·*IP* from OBDD is in OutputP [Madre and Coudert, 1991], and it is almost straightforward to extend this result to dec-DNNF. To make it precise, let us briefly describe the output polynomial construction of *IP*(Σ) for Σ, a dec-DNNF circuit. The construction is based on the three following, folklore propositions (for the sake of completeness, a proof for each of them is nonetheless reported in the full-proof version of the paper).

Proposition 3. *Let f and g be Boolean functions, then IP$(f \wedge g) = \max(\{t \wedge t' \mid t \in IP(f), t' \in IP(g)\}, \models)$. Furthermore if $var(f) \cap var(g) = \emptyset$, then IP$(f \wedge g) = \{t \wedge t' \mid t \in IP(f), t' \in IP(g)\}$.*

Proposition 4. *Let f a Boolean function, let x be a variable, and let $\ell \in \{x, \overline{x}\}$. Consider $t \in IP(f|\ell)$. If $t \models f|\overline{\ell}$, then $t \in IP(f)$, otherwise $t \wedge \ell \in IP(f)$.*

Proposition 5. *Let f be a Boolean function and let x be a variable.*

$$\begin{aligned} IP(f) = &\{t \wedge \overline{x} \mid t \in IP(f|\overline{x}),\ t \not\models f|x\} \\ &\cup \{t \wedge x \mid t \in IP(f|x),\ t \not\models f|\overline{x}\} \\ &\cup IP(f|\overline{x} \wedge f|x) \end{aligned}$$



(a) A dec-DNNF circuit (b) A path followed in `MissingIP` (c) Propagation of an implicant

Figure 1: Generation of a new prime implicant from a dec-DNNF circuit

Note that $t \in IP(f|\overline{x})$ (resp. $IP(f|x)$) entails $f|x$ (resp. $f|\overline{x}$) if and only if t is subsumed by some term in $IP(f|\overline{x} \wedge f|x)$. As a consequence, from $IP(f|\overline{x})$ and $IP(f|x)$, one can construct $IP(f|\overline{x} \wedge f|x)$ in polynomial time thanks to Proposition 3 and we use it to derive $IP(f)$ thanks to Proposition 5.

We also have that (see the algorithm for conditioning a prime implicant representation provided in [Darwiche and Marquis, 2002]):

Proposition 6. *Let f a Boolean function and let x be a variable, then $|IP(f)| \geq \max(|IP(f|\overline{x})|, |IP(f|x)|)$.*

Consider now a dec-DNNF circuit Σ and an internal node v with two children u and w. If the sets $IP(\Sigma_u)$ and $IP(\Sigma_w)$ are provided, then $IP(\Sigma_v)$ is obtained in polynomial time using Proposition 3 if v is a decomposable $\wedge$-gate, and using Proposition 5 if v is a decision node. Furthermore, in both cases, we have $|IP(\Sigma_v)| \geq \max(|IP(\Sigma_u)|, |IP(\Sigma_w)|)$. These observations lead to a simple algorithm that generates $IP(\Sigma)$ by computing the sets $IP(\Sigma_v)$ for every node v of Σ considered in a bottom-up way. Since constructing the set of prime implicants for any node given that of its children is tractable, since this set is smaller than $|IP(\Sigma)|$, and since it is computed only once, the algorithm runs in time $O(poly(|\Sigma| + |IP(\Sigma)|))$. Thus, we get:

Proposition 7. *Enum·IP from dec-DNNF is in* OutputP.

Example 1. We give the construction of the sets of prime implicants for the nodes v_1, v_2, v_3 in the dec-DNNF circuit Σ represented on Figure 1b.

- v_3: the sets of prime implicants of the children are $IP(1) = \{t_\emptyset\}$ and $IP(p) = \{p\}$. Using Proposition 5 we have that $\overline{s} \wedge t_\emptyset = \overline{s}$ and $\Sigma_{v_3}|s = p$, so $\overline{s} \wedge t_\emptyset \not\models \Sigma_{v_3}|s$ showing that $\overline{s} \in IP(\Sigma_{v_3})$. We also have that $\Sigma_{v_3}|\overline{s} = 1$, so $s\,p \models \Sigma_{v_3}|\overline{s}$ showing that $s\,p \notin IP(\Sigma_{v_3})$. Finally, we have that $IP(\Sigma_{v_3}|s \wedge \Sigma_{v_3}|\overline{s}) = \{p\}$ by Proposition 3, so $IP(\Sigma_{v_3}) = \{\overline{s}, p\}$.
- v_2: the sets of prime implicants of the children are $IP(\overline{p}) = \{\overline{p}\}$ and $IP(\Sigma_{v_3})$ so we compute $IP(\Sigma_{v_2}) = \{b\,\overline{p}, \overline{b}\,\overline{s}, \overline{b}\,p, \overline{p}\,\overline{s}\}$
- v_1: the sets of prime implicants of the children are $IP(\overline{p} \wedge \overline{s}) = \{\overline{p}\,\overline{s}\}$ and $IP(\Sigma_{v_2})$ so we compute $IP(\Sigma_{v_1}) = \{\overline{e}\,\overline{b}\,p, \overline{e}\,b\,\overline{p}, \overline{e}\,\overline{b}\,\overline{s}, \overline{p}\,\overline{s}\}$

4 Enum·IP from Dec-DNNF is in IncP

We now investigate Enum·IP from dec-DNNF from the incremental enumeration perspective. Based on Proposition 1, we design a tractable algorithm `AnotherIP` for solving the problem AnotherSol·*IP*, thus showing that Enum·*IP* from dec-DNNF is in IncP.

4.1 Solving the Decision Variant of AnotherSol·IP

We first consider the decision variant of AnotherSol·*IP* from dec-DNNF: given a dec-DNNF circuit Σ and a set $S \subseteq IP(\Sigma)$, return *false* if and only if $S \neq IP(\Sigma)$. Recall from the discussion preceding Proposition 7 that there is a bottom-up procedure for generating all prime implicants of the dec-DNNF circuit Σ. To address the decision variant of AnotherSol·*IP* on inputs Σ and S, a reverse, top-down search is performed, assuming that S is $IP(\Sigma)$ until finding a contradiction.

Before defining what a contradiction means in this setting, a few notations are useful. For t a term and X a set of variables, t_X denotes the restriction of t to variables in X. Note that if X and $var(t)$ are disjoint, then t_X is the empty term $t_\emptyset$.

Proposition 8. *Let Σ be a dec-DNNF circuit and let $S \subseteq IP(\Sigma)$. If the root of Σ is an $\wedge$-node, let u and w be its children and let $S_u = \{t_{var(\Sigma_u)} \mid t \in S\}$ and $S_w = \{t_{var(\Sigma_w)} \mid t \in S\}$. Then $S_u \subseteq IP(\Sigma_u)$ and $S_w \subseteq IP(\Sigma_w)$ hold, and*

$$S = IP(\Sigma) \text{ iff } S_u = IP(\Sigma_u) \text{ and } S_w = IP(\Sigma_w).$$

Proposition 9. *Let Σ be a dec-DNNF circuit whose root is a decision node labelled by x. Let u be its 0-child and w be its 1-child. Given $S \subseteq IP(\Sigma)$, let $S_u = \{t \mid t \wedge \overline{x} \in S\} \cup (S \cap IP(\Sigma_u))$, $S_w = \{t \mid t \wedge x \in S\} \cup (S \cap IP(\Sigma_w))$ and $S' = \{t \mid t \in S, x \notin var(t)\}$. Then $S_u \subseteq IP(\Sigma_u)$ and $S_w \subseteq IP(\Sigma_w)$ hold, and*

$$S = IP(\Sigma) \text{ iff } S_u = IP(\Sigma_u) \text{ and } S_w = IP(\Sigma_w) \text{ and } S' = \max(\{t_u \wedge t_w \mid t_u \in S_u,\, t_w \in S_w\}, \models).$$

Let v be the root of the dec-DNNF circuit Σ and let $S \subseteq IP(\Sigma)$. We say that we have a *contradiction* at node v when

(c1) $S = \emptyset$ while Σ is satisfiable, or

(c2) v is a decision node, $S_u = IP(\Sigma_u)$ and $S_w = IP(\Sigma_w)$, but $S' \neq \max(\{t_u \wedge t_w \mid t_u \in S_u,\, t_w \in S_w\}, \models)$.

Algorithm 1: `MissingIP`(Σ, S, P)

Promises: Σ is reduced, $S \subseteq IP(\Sigma)$
Let v be the root of Σ and let $P' \leftarrow P \cup (v)$
if $\lambda(v) = |S|$ **then** return *false*
if $S = \emptyset$ **then**
  if v is labelled by 0 **then** set $\lambda(v)$ to 0, return *false*
  else return (`GenerateIP`(Σ), P')
end
if v *is a* $\wedge$-node with children u and w **then**
  Build S_u and S_w as in Proposition 8
  $r \leftarrow$ `MissingIP`(Σ_u, S_u, P')
  if $r \neq$ *false* **then** return r
  $r \leftarrow$ `MissingIP`(Σ_w, S_w, P')
  if $r \neq$ *false* **then** return r
else if v is a decision node with children u and w **then**
  Build S_u, S_w, S' as in Proposition 9
  $r \leftarrow$ `MissingIP`(Σ_u, S_u, P')
  if $r \neq$ *false* **then** return r
  $r \leftarrow$ `MissingIP`(Σ_w, S_w, P')
  if $r \neq$ *false* **then** return r
  $S^* \leftarrow \max(\{t_u \wedge t_w \mid t_u \in S_u, t_w \in S_w\}, \models)$
  if $S^* \neq S'$ **then** for any $t \in S^* \setminus S'$ return (t, P')
end
Set $\lambda(v)$ to $|S|$ and return *false*

A contradiction guarantees that $S \neq IP(\Sigma)$. The contradiction (c1) is easy to check. Contradiction (c2) on the other hand requires to show that $S_u = IP(\Sigma_u)$ and $S_w = IP(\Sigma_w)$. When v is an internal node, with children u and w, if there is no contradiction (c1) at v, we use Propositions 8 and 9 to build from S two sets S_u and S_w that we recursively compare to $IP(\Sigma_u)$ and $IP(\Sigma_w)$. Either the recursion ends under u or w on a contradiction, in which case $S \neq IP(\Sigma)$, or it stops by itself (i.e., when reaching the leaves of the circuit), which shows that $S_u = IP(\Sigma_u)$ and $S_w = IP(\Sigma_w)$, and then we can check whether there is contradiction (c2) at node v. If there is none, then $S = IP(\Sigma)$.

The procedure is given by Algorithm `MissingIP`. The inputs are a dec-DNNF circuit Σ, a set $S \subseteq IP(\Sigma)$ and a path P in Σ (which will be useful later). A function λ mapping the nodes of Σ to integers is used for memoization purposes. Initially $\lambda(v) = -1$ for every node v, but $\lambda(v)$ may be assigned a non-negative value at some point. More precisely, the first time a call `MissingIP`(Σ_v, S, P) returns *false*, we learn that $S = IP(\Sigma_v)$ and set $\lambda(v)$ to $|S|$. Then for each later call `MissingIP`(Σ_v, S', P') with $S' \subseteq IP(\Sigma_v)$, we check whether $S' = IP(\Sigma_v)$ by verifying that $\lambda(v) = |S'|$.

Proposition 10. *Given a reduced dec-DNNF circuit Σ and $S \subseteq IP(\Sigma)$,* `MissingIP`$(\Sigma, S, \emptyset)$ *runs in time $O(poly(|S| + |\Sigma|))$, and it returns false if and only if $S = IP(\Sigma)$.*

4.2 Augmenting an Incomplete Subset of IP(Σ)

We build upon `MissingIP` so that, when $S \neq IP(\Sigma)$, we also return a prime implicant in $IP(\Sigma) \setminus S$. The idea is to use the path P to keep track of the ancestor nodes that were visited before reaching a contradiction and to use P to construct a prime implicant in $IP(\Sigma) \setminus S$. As an example, consider calling `MissingIP`$(\Sigma, S_0, \emptyset)$ with Σ the dec-DNNF circuit of Figure 1a and $S_0 = \{h\,\overline{e}\,b\,\overline{p},\ h\,\overline{p}\,\overline{s},\ \overline{e}\,\overline{p}\,\overline{s}\}$ a set of prime implicants of Σ. Figure 1b shows a scenario when `MissingIP`$(\Sigma, S_0, \emptyset)$ calls `MissingIP`$(\Sigma_{v_1}, S_1, (v_0))$, which calls in turn`MissingIP`$(\Sigma_{v_2}, S_2, (v_0, v_1))$, which finally calls `MissingIP`$(\Sigma_{v_3}, S_3, (v_0, v_1, v_2))$. Since $S_3 = \emptyset$ and Σ_{v_3} is reduced and different from 0, the algorithm has reached a contradiction (c1) at node v_3 and has not returned *false*, thus indicating that $S_0 \neq IP(\Sigma)$. `MissingIP` has followed the path $P = (v_0, v_1, v_2, v_3)$ to reach that contradiction and has kept it in memory. This path P can then be used to generate a prime implicant in $IP(\Sigma) \setminus S_0$. First `MissingIP` returns the path P to v_3 as well as a prime implicant of Σ_{v_3}, say it is $\overline{s}$. Then we construct a prime implicant of Σ_{v_2} upon $\overline{s}$, here since v_3 is the 0-child of v_2 and since $\overline{s}$ does not entail the 1-child of v_2 we obtain $\overline{b}\,\overline{s} \in IP(\Sigma_{v_2})$. Then we construct a prime implicant of Σ_{v_1} upon $\overline{b}\,\overline{s}$, here since v_2 is the 0-child of v_1 and since $\overline{b}\,\overline{s}$ does not entail the 1-child of v_1 we obtain $\overline{e}\,\overline{b}\,\overline{s} \in IP(\Sigma_{v_1})$. Repeating the step one more time leads to $h\,\overline{e}\,\overline{b}\,\overline{s} \in IP(\Sigma_{v_0}) = IP(\Sigma)$. The procedure is illustrated in Figure 1c. In this example, for generating a new prime implicant of Σ, we have created $t \in \Sigma_{v_3} \setminus S_3$ and augmented it using Proposition 4 as we travelled backwards along P. We say that we have *propagated* t along the path P.

Algorithm 2: `GenerateIP`(Σ)

Promise: Σ is satisfiable
Find a satisfying assignment a of Σ
Let $t = \bigwedge_{a(x)=1} x \wedge \bigwedge_{a(x)=0} \overline{x}$
while there is $\ell \in t$ such that $t - \ell \models \Sigma$ **do**
  Remove ℓ from t
end
Return t

Accordingly, the algorithm `AnotherIP` to generate a new prime implicant breaks into two steps. First `MissingIP`(Σ, S, P) searches for a contradiction. It returns $false$ if $S = IP(\Sigma)$ or a pair (t, P) with P the path followed to reach a node v where a contradiction has been found (like v_3 in the example), and t a prime implicant of Σ_v that could not be derived from S. The procedure `GenerateIP` is used to generate t. `GenerateIP` runs in polynomial time thanks to linear-time implicant check on dec-DNNF circuits. Finally `PropagateIP` is called to propagate t along the path P.

The next proposition shows the correctness of `AnotherIP`:

Proposition 11. *Let Σ be a reduced dec-DNNF circuit and let $S \subseteq IP(\Sigma)$.* `AnotherIP`(Σ, S) *runs in time $O(poly(|S| + |\Sigma|))$. It returns false if $S = IP(\Sigma)$, otherwise it returns a prime implicant of Σ that does not belong to in S.*

On this basis, the existence of a polynomial incremental time enumeration of prime implicants for dec-DNNF circuits can be easily established:

Proposition 12. *Enum·IP from dec-DNNF is in* IncP.

Algorithm 3: `Propagate`$(\Sigma, t, P = (v_0, \ldots, v_i))$

Promise: Σ is reduced, its root is v_0, P is a path in Σ
if $|P| = 1$ **then** return t
if v_{i-1} is a $\wedge$-node with children u and w **then**
  if $v_i = u$ **then** $t' \leftarrow$ `GenerateIP`(Σ_w)
  if $v_i = w$ **then** $t' \leftarrow$ `GenerateIP`(Σ_u)
else if v_{i-1} is a decision node for variable x with 0-child u and 1-child w **then**
  if $v_i = u$ **then**
    if $t \models \Sigma_w$ **then** $t' \leftarrow t_\emptyset$ **else** $t' \leftarrow \overline{x}$
  else
    if $t \models \Sigma_u$ **then** $t' \leftarrow t_\emptyset$ **else** $t' \leftarrow x$
end
`Propagate`$(\Sigma, t \wedge t', (v_0, \ldots, v_{i-1}))$

Algorithm 4: `AnotherIP`(Σ, S)

Promise: Σ is reduced, $S \subseteq IP(\Sigma)$
$r \leftarrow$ `MissingIP`$^*(\Sigma, S, \emptyset)$
if $r = false$ **then** return $false$
else if $r = (t, P)$ **then** return `Propagate`(Σ, t, P)

5 Enumerating Specific Prime Implicants

For some applications, enumerating all prime implicants of f makes sense, even though there can be exponentially many. We have already mentioned the dualization of monotone CNF formulae as an example. In this section, we describe two problems that ask for generating only specific prime implicants, representing respectively subset-minimal abductive explanations and sufficient reasons.

To illustrate the two notions we use the function f computed by the dec-DNNF circuit of Figure 1a as a toy example. f encodes a very incomplete characterization of human-like creatures in Tolkien's Middle Earth based on four physical attributes: presence of beard and facial hair (b), small size (s), human-like skin (h), pointy ears (p), plus the indication of whether the creature is enrolled in the armies of evil (e). We imagine that there are only seven possible creatures: hobbits ($h\,\overline{b}\,p\,s\,\overline{e}$), elves ($\overline{h}\,\overline{b}\,p\,\overline{s}\,\overline{e}$), dwarfs ($h\,b\,\overline{p}\,s\,\overline{e}$), men and women ($h*\overline{p}\,\overline{s}\,*$),[1] ents ($\overline{h}*\overline{p}\,\overline{s}\,\overline{e}$), orcs ($\overline{h}\,\overline{b}\,p*e$) and trolls ($\overline{h}\,\overline{b}\,\overline{p}\,\overline{s}\,e$). The satisfying assignments of f describe these creatures. Its prime implicants are the smallest combinations of attributes which guarantee the existence of a creature in our Middle Earth.

5.1 Abductive Explanations

Abductive explanations (see e.g., [Selman and Levesque, 1990; Eiter and Gottlob, 1995]) can be defined as follows:

Definition 1 (**Abductive explanation**)**.** Given a Boolean function f over variables X, a subset $H \subseteq X$, and a term m on $X \setminus H$, an *abductive explanation* is a term t on H such that $f \wedge t$ is satisfiable and $f \wedge t \models m$.

[1] $*$ denotes that both choices are possible for the variable, typically here humans may fight for evil, humans and ents may or may not have beards, and orcs have a wide range of size.

The abduction problem asks whether an abductive explanation t exists for the input (f, H, m).

Example 2. Consider our toy example. We look for combinations of physical attributes that guarantee that the creature is evil. This is an abduction problem with $H = \{h, b, p, s\}$ and $m = e$. For instance the term $\overline{h} \wedge p$ is an abductive explanation because there exist creatures with pointy ears and a skin that are not human-like, and all of them are evil (in this case only the orcs fit the description).

It is easy to see that an abductive explanation t is in fact an implicant of $\neg f \vee m$ with the conditions that $f \wedge t$ is satisfiable and that t is restricted to variables in H (the abducibles). Furthermore, since abduction is not a truth-preserving form of inference, one is often interested in generating subset-minimal abductive explanations only (i.e., the logically weakest abductive explanations); they correspond to the *prime* implicants of $\neg f \vee m$ such that $f \wedge t$ is satisfiable and t is restricted to variables in H.

Obviously enough, the abduction problem we focus on (the existence of an abductive explanation) is the same, would we consider subset-minimal abductive explanations or not. Indeed, deciding whether an abductive explanation exists is equivalent to deciding whether a subset-minimal abductive explanation exists. Unfortunately, the condition that only variables in H are allowed in abductive explanations is already too demanding from an enumeration perspective.

Proposition 13. *Unless* $\mathsf{P} = \mathsf{NP}$*, there is no polynomial-time algorithm which, given an OBDD circuit or a decision tree computing a function f over X and a set $Y \subseteq X$, decides whether f has an implicant t with $var(t) \subseteq Y$.*

5.2 Sufficient Reasons

The notion of sufficient reason[2] [Darwiche and Hirth, 2020] (aka prime implicant explanation [Shih *et al.*, 2018b]) is defined as follows:

Definition 2 (**Sufficient reason**)**.** Given a Boolean function f, let a be any assignment to a superset of $var(f)$. A *sufficient reason* for a is a prime implicant t of f (resp. $\neg f$) such that a satisfies t, provided that a satisfies f (resp. $\neg f$). The set of all sufficient reasons for a given f is denoted by $SR(f, a)$ (resp. $SR(\neg f, a)$) when a satisfies f (resp. $\neg f$).

Example 3. Consider again our toy example. There is no creature which is small, has human-like skin, pointy ears, no facial hair, and is evil. Finding the reasons of why such a creature cannot exist means finding sufficient reasons for the assignment a defined by $a(h) = a(p) = a(s) = a(e) = 1$ and $a(b) = 0$ given $\neg f$. In this case $h\,p\,e \in SR(\neg f, a)$ explains why such a creature cannot exist: there are no creatures that are evil and have both human-like skin and pointy ears, but there are such creatures that are non-evil (hobbits and elves), and there are evil creatures that have pointy ears (orcs) or human-like skin (men). There are other sufficient reasons for a given $\neg f$, for instance $h\,s\,e \in SR(\neg f, a)$.

[2] This concept is also referred to as "abductive explanations" [Ignatiev *et al.*, 2019; Ignatiev *et al.*, 2020]; in the following, we stick to "sufficient reason" to avoid any confusion with the (distinct) concept of abductive explanations as discussed in the previous section.

We define the problem Enum·*SR* similarly to Enum·*IP*. A couple of results about the complexity of computing sufficient reasons have been pointed out for the past few years. Obviously enough, when no assumption is made on the representation of f, computing a single sufficient reason for an assignment a is already NP-hard (for pretty much the same reasons as for the prime implicant case, i.e., f is valid iff for any a, the unique sufficient reason for a given f is the empty term). Furthermore, the number of sufficient reasons for an assignment a given f can be exponential in the number of variables even when f is represented in DT [Audemard *et al.*, 2021]. Contrary to abductive explanations, it is computationally easy to generate a single sufficient reason from $SR(\Sigma, a)$ when Σ is an OBDD circuit or a decision tree representing f. A greedy algorithm can be used to this end: if a satisfies Σ (resp. $\neg\Sigma$), then start with the canonical term having a as its unique satisfying assignment and remove literals from this term while ensuring that it still is an implicant of Σ (resp. $\neg\Sigma$), until no more literal can be removed. In addition, when Σ is in DT, we can generate in polynomial time a monotone CNF formula Ψ such that $IP(\Psi) = SR(\Sigma, a)$ (see [Darwiche and Marquis, 2021] for details), and then take advantage of a quasi-polynomial time algorithm for enumerating the elements of $IP(\Psi)$ [Gurvich and Khachiyan, 1999]. Contrastingly, deciding whether a preset number of sufficient reasons for a given a exists is intractable (NP-hard), even when the Boolean function f is monotone (see Theorem 3 in [Marques-Silva *et al.*, 2021]).

In the following, we complete those results by providing evidence that Enum·SR from any language among dec-DNNF, OBDD, or DT is a difficult problem, despite the fact that those languages are quite convenient for many reasoning tasks [Darwiche and Marquis, 2002; Koriche *et al.*, 2013].

Let us first give an inductive computation of $SR(\Sigma, a)$ similar to that of $IP(\Sigma)$.

Proposition 14. *Let f and g be Boolean functions with $var(f) \cap var(g) = \emptyset$ and let a be a truth assignment to a superset of $var(f) \cup var(g)$, then $SR(f \wedge g, a) = \{t \wedge t' \mid t \in SR(f, a), t' \in SR(g, a)\}$.*

Proposition 15. *Let f be a Boolean function, let a be a truth assignment to a superset of $var(f)$ and let $x \in var(f)$. If a satisfies the literal ℓ on variable x then*

$$SR(f, a) = \{t \wedge \ell \mid t \in SR(f|\ell, a),\ t \not\models f|\overline{\ell}\} \cup SR(f|\overline{x} \wedge f|x, a).$$

By Proposition 6, $|IP(f)| \geq \max(|IP(f|\overline{x})|, |IP(f|x)|)$. In a sense this means that using $IP(f|\overline{x})$ and $IP(f|x)$ to generate $IP(f)$ is not a waste of resources since all these implicants are kept in some form through $IP(f)$. This led to our output polynomial procedure to generate $IP(f)$ for OBDD and more generally for dec-DNNF circuits. On the other hand, it is not guaranteed that $SR(f, a)$ is larger than $SR(f|x, a)$ and $SR(f|\overline{x}, a)$ so there is no straightforward adaptation of this procedure from Enum·IP to Enum·SR.

Example 4. Let Σ be the dec-DNNF circuit of Figure 1a. Consider the dec-DNNF circuit Σ_{v_1} rooted at node v_1, as spotted in Figure 1b. The assignment a to $\{b, e, p, s\}$ defined by $a(b) = a(e) = 1$ and $a(p) = a(s) = 0$ satisfies Σ_{v_1}. Recall that the set $IP(\Sigma_{v_1})$ has been constructed in Example 1 and observe that $SR(\Sigma_{v_1}, a) = \{\overline{p}\,\overline{s}\}$. Now the 0-child of v_1 is v_2 and looking at the set $IP(\Sigma_{v_2})$ constructed in Example 1, we see that $SR(\Sigma_{v_2}, a) = \{\overline{p}\,\overline{s}, b\,\overline{p}\}$. Since $\Sigma_{v_2} = \Sigma_{v_1}|\overline{e}$, we have that $|SR(\Sigma_{v_1}, a)| < \max(|SR(\Sigma_{v_1}|\overline{e}, a)|, |SR(\Sigma_{v_1}|e, a)|)$.

Actually, we give evidence that enumerating sufficient reasons from dec-DNNF, and even from OBDD or DT, is not in OutputP by reducing to it the problem of enumerating the minimal transversals of a hypergraph, a well-known problem whose membership to OutputP is a long-standing question. Formally:

Proposition 16. *If Enum·SR from OBDD is in* OutputP *or Enum·SR from DT is in* OutputP*, then enumerating the minimal transversals of a hypergraph is in* OutputP.

6 Conclusion

Most applications of prime implicants for Boolean function analysis use only a fraction of the many prime implicants a Boolean function may have. Especially, in the context of logic-based abduction, subset-minimal assumptions to be added to the available background knowledge in order to be able to derive some given manifestations are looked for; in the propositional case, they correspond to specific prime implicants. Furthermore, in an eXplainable AI perspective, specific prime implicants known as sufficient reasons are used to explain the predictions of machine learning algorithms.

In our work, we have studied the enumeration of general and specific prime implicants of Boolean functions represented as dec-DNNF circuits. It was known that these circuits enable efficient reasoning on Boolean functions. Our results show that when it comes to prime implicants enumeration, dec-DNNF circuits have benefits as well as limitations. Our take-home message is that, while dec-DNNF circuits enable enumerating general prime implicants in incremental polynomial time, there are strong pieces of evidence against the existence of any output-polynomial time procedure for enumerating specific prime implicants from dec-DNNF circuits. More precisely, if a procedure for enumerating subset-minimal abductive explanations were to exist, then P = NP would follow. Similarly, if there were an output-polynomial time algorithm for enumerating sufficient reasons from dec-DNNF circuits, then the enumeration of the minimal transversals of a hypergraph would be in OutputP. Though this is considered unlikely in enumeration complexity, we think that proving a stronger statement would be a valuable contribution. We let this task open for future research.

Acknowledgments

Many thanks to the anonymous reviewers for their comments and insights. This work has benefited from the supports of the PING/ACK project (ANR-18-CE40-0011) and of the AI Chair EXPEKCTATION (ANR-19-CHIA-0005-01) of the French National Research Agency. It was also partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215.

References

[Audemard *et al.*, 2021] Gilles Audemard, Steve Bellart, Louenas Bounia, Frédéric Koriche, Jean-Marie Lagniez, and Pierre Marquis. On the explanatory power of decision trees. *CoRR*, abs/2108.05266, 2021.

[Bryant, 1986] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–692, 1986.

[Chandra and Markowsky, 1978] Ashok K. Chandra and George Markowsky. On the number of prime implicants. *Discrete Mathematics*, 24:7–11, 1978.

[Darwiche and Hirth, 2020] Adnan Darwiche and Auguste Hirth. On the reasons behind decisions. In *Proc. of ECAI'20*, pages 712–720, 2020.

[Darwiche and Marquis, 2002] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17:229–264, 2002.

[Darwiche and Marquis, 2021] Adnan Darwiche and Pierre Marquis. On quantifying literals in Boolean logic and its applications to explainable AI. *J. Artif. Intell. Res.*, 72:285–328, 2021.

[Darwiche, 2001] Adnan Darwiche. Decomposable negation normal form. *Journal of the Association for Computing Machinery*, 48(4):608–647, 2001.

[de Kleer, 1986] Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–167, 1986.

[Dunham and Fridshal, 1959] Bradford Dunham and Richard Fridshal. The problem of simplifying logical expressions. *Journal of Symbolic Logic*, 1959.

[Eiter and Gottlob, 1995] Thomas Eiter and Georg Gottlob. The complexity of logic-based abduction. *Journal of the ACM*, 42(1):3–42, 1995.

[Eiter *et al.*, 2008] Thomas Eiter, Kazuhisa Makino, and Georg Gottlob. Computational aspects of monotone dualization: A brief survey. *Discret. Appl. Math.*, 156(11):2035–2049, 2008.

[Gergov and Meinel, 1994] Jordan Gergov and Christoph Meinel. Efficient analysis and manipulation of OBDDs can be extended to FBDDs. *IEEE Transactions on Computers*, 43(10):1197–1209, 1994.

[Gurvich and Khachiyan, 1999] Vladimir Gurvich and Leonid Khachiyan. On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions. *Discret. Appl. Math.*, 96-97:363–373, 1999.

[Ignatiev *et al.*, 2019] Alexey Ignatiev, Nina Narodytska, and João Marques-Silva. Abduction-based explanations for machine learning models. In *Proc. of AAAI'19*, pages 1511–1519, 2019.

[Ignatiev *et al.*, 2020] Alexey Ignatiev, Nina Narodytska, Nicholas Asher, and João Marques-Silva. On relating 'why?' and 'why not?' explanations. *CoRR*, abs/2012.11067, 2020.

[Koriche *et al.*, 2013] Frédéric Koriche, Jean-Marie Lagniez, Pierre Marquis, and Samuel Thomas. Knowledge compilation for model counting: Affine decision trees. In *Proc. of IJCAI'13*, pages 947–953, 2013.

[Madre and Coudert, 1991] Jean Christophe Madre and Olivier Coudert. A logically complete reasoning maintenance system based on a logical constraint solver. In *Proc. of IJCAI'91*, pages 294–299, 1991.

[Marques-Silva *et al.*, 2021] João Marques-Silva, Thomas Gerspacher, Martin C. Cooper, Alexey Ignatiev, and Nina Narodytska. Explanations for monotonic classifiers. In *Proc. of ICML'21*, pages 7469–7479, 2021.

[Marquis, 2000] Pierre Marquis. *Consequence finding algorithms*, volume 5 of *Handbook on Defeasible Reasoning and Uncertainty Management Systems*, chapter 2, pages 41–145. Kluwer Academic Publisher, 2000.

[Miller, 2019] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, 2019.

[Molnar, 2019] Christoph Molnar. *Interpretable Machine Learning - A Guide for Making Black Box Models Explainable*. Leanpub, 2019.

[Narodytska *et al.*, 2018] Nina Narodytska, Shiva Prasad Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. In *Proc. of AAAI'18*, pages 6615–6624, 2018.

[Oztok and Darwiche, 2014] Umut Oztok and Adnan Darwiche. On compiling CNF into Decision-DNNF. In *Proc. of CP'14*, pages 42–57, 2014.

[Quine, 1952] Willard Van Orman Quine. The problem of simplifying truth functions. *American Mathematical Monthly*, 59:521–531, 1952.

[Reiter and de Kleer, 1987] Raymond Reiter and Johan de Kleer. Foundations of assumption-based truth maintenance systems: preliminary report. In *Proc. of AAAI'87*, pages 183–188, 1987.

[Selman and Levesque, 1990] Bart Selman and Hector J. Levesque. Abductive and default reasoning: a computational core. In *Proc. of AAAI'90*, pages 343–348, 1990.

[Shih *et al.*, 2018a] Andy Shih, Arthur Choi, and Adnan Darwiche. Formal verification of Bayesian network classifiers. In *Proc. of PGM'18*, pages 427–438, 2018.

[Shih *et al.*, 2018b] Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining Bayesian network classifiers. In *Proc. of IJCAI'18*, pages 5103–5111, 2018.

[Shih *et al.*, 2019] Andy Shih, Arthur Choi, and Adnan Darwiche. Compiling Bayesian networks into decision graphs. In *Proc. of AAAI'19*, pages 7966–7974, 2019.

[Strozecki, 2019] Yann Strozecki. Enumeration complexity. *Bull. EATCS*, 129, 2019.

[Wegener, 2000] Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000.