Efficient Explanations for Knowledge Compilation Languages

Xuanxiang Huang, Yacine Izza, Alexey Ignatiev, Martin Cooper, Nicholas Asher, Joao Marques-Silva

Efficient Explanations for Knowledge Compilation Languages

Xuanxiang Huang
Université de Toulouse, Toulouse, France

Yacine Izza
Université de Toulouse, Toulouse, France

Alexey Ignatiev
Monash University, Melbourne, Australia

Martin C. Cooper
Université Paul Sabatier, IRIT, Toulouse, France

Nicholas Asher
IRIT, CNRS, Toulouse, France

Joao Marques-Silva
IRIT, CNRS, Toulouse, France

Abstract

Knowledge compilation (KC) languages find a growing number of practical uses, including in Constraint Programming (CP) and in Machine Learning (ML). In most applications, one natural question is how to explain the decisions made by models represented by a KC language. This paper shows that for many of the best known KC languages, well-known classes of explanations can be computed in polynomial time. These classes include deterministic decomposable negation normal form (d-DNNF), and so any KC language that is strictly less succinct than d-DNNF. Furthermore, the paper also investigates the conditions under which polynomial time computation of explanations can be extended to KC languages more succinct than d-DNNF.

2012 ACM Subject Classification Theory of computation → Automated reasoning

Keywords and phrases Machine Learning, Explainable AI, Knowledge Compilation, Tractability

Funding This work was supported by the AI Interdisciplinary Institute ANITI, funded by the French program "Investing for the Future – PIA3" under Grant agreement no. ANR-19-PI3A-0004, and by the H2020-ICT38 project COALA "Cognitive Assisted agile manufacturing for a Labor force supported by trustworthy Artificial intelligence".

1 Introduction

The growing use of machine learning (ML) models in practical applications raises a number of concerns related with fairness, robustness, but also explainability [35, 59, 41]. Recent years have witnessed a number of works on computing explanations for the predictions made by ML models[1]. Approaches to computing explanations can be broadly categorized as heuristic [47, 36, 48], which offer no formal guarantees of rigor, and non-heuristic [52, 27, 15, 4], which in contrast offer strong guarantees of rigor. Non-heuristic explanation approaches can be further categorized into compilation-based [52, 53, 15] and oracle-based [27, 37].

Compilation-based approaches resort to knowledge compilation (KC) languages, often to compile the decision function associated with an ML classifier [52, 53]. As a result, more recent work studied KC languages from the perspective of explainability, with the purpose of

[1] There is a fast growing body of work on the explainability of ML models. Example references include [23, 49, 50, 39, 38, 2, 40, 60, 42].

understanding the complexity of computing explanations [4, 6, 3] but also with the goal of identifying examples of queries of interest [4, 3]. Observe that besides serving to compile the decision function of some classifier, functions represented with KC languages can also be viewed as classifiers. In addition, explanations for the behavior of functions expressed in KC languages find applications other than explaining ML models, including explanations in constraint programming [1, 7, 17, 8, 21]. Furthermore, although recent work [4, 6, 3] analyzed the complexity of explainability queries for different KC languages, it is also the case that it is unknown which KC languages allow the expressible functions to be explained efficiently, and which do not. On the one hand, [4, 3] proposes conditions not met by most KC languages. On the other hand [6] studies restricted cases of KC languages, but focusing on smallest PI-explanations. Also, since one key motivation for the use of KC languages is the efficiency of reasoning, namely with respect to specific queries and transformations [16], a natural question is whether similar results can be obtained in the setting of explainability.

This paper studies the computational complexity of computing PI-explanations [52] and contrastive explanations [39] for classifiers represented with KC languages. Concretely, the paper shows that for any KC language that implements in polynomial time the well-known queries of consistency (**CO**) and validity (**VA**), and the transformation of conditioning (**CD**), then one PI-explanation or one contrastive explanation can be computed in polynomial time. This requirement is strictly less stringent than another one proposed in earlier work [4]. As a result, for a large number of KC languages, that include d-DNNF, one PI-explanation or one contrastive explanation can be computed in polynomial time. The result immediately generalizes to KC languages less succinct than d-DNNF, e.g. OBDD, SDD, to name a few. Moreover, for the concrete case of SDDs, the paper shows that practical optimizations lead to clear performance gains. Besides computing one explanation, one is often interested is obtained multiple explanations, thus allowing a decision maker to get a better understanding of the reasons supporting a decision. As a result, the paper also proposes a MARCO-like [34] algorithm for the enumeration of both AXps and CXps. Furthermore, the paper studies the computational complexity of explaining generalizations of decision sets [32], and proposes conditions under which explanations can be computed in polynomial time. Finally, the paper studies multi-class classifiers, and again proposes conditions for finding explanations in polynomial time.

The paper is organized as follows. Section 2 introduces the definitions and notation used throughout the paper. Section 3 shows that for a large class of KC languages, one PI-explanation and one contrastive explanation can be computed in polynomial time. Concretely, the paper shows that d-DNNF can be explained in polynomial time, and so any less succinct language can also be explained in polynomial time. Furthermore, the paper shows that sentential decision diagrams (SDDs) enable practical optimizations that yield more efficient algorithms in practice. In addition, Section 3 shows how to enumerate explanations requiring one NP oracle call for each computed explanation. Section 4 investigates a number of generalized classifiers, which can be built from KC languages used as building blocks. Section 5 assesses the computation of explanations of d-DNNF's and SDDs in practical settings. Section 6 concludes the paper.

2 Preliminaries

Classification problems & formal explanations. This paper considers classification problems, which are defined on a set of features (or attributes) $\mathcal{F} = \{1, \ldots, m\}$ and a set of classes $\mathcal{K} = \{c_1, c_2, \ldots, c_K\}$. Each feature $i \in \mathcal{F}$ takes values from a domain $\mathbb{D}_i$. In general,

domains can be boolean, integer or real-valued, but in this paper we restrict $\mathbb{D}_i = \{0, 1\}$ and $\mathcal{K} = \{0, 1\}$. (In the context of KC languages, we will replace 0 by $\bot$ and 1 by $\top$. This applies to domains and classes.) Feature space is defined as $\mathbb{F} = \mathbb{D}_1 \times \mathbb{D}_2 \times \ldots \times \mathbb{D}_m = \{0, 1\}^m$. The notation $\mathbf{x} = (x_1, \ldots, x_m)$ denotes an arbitrary point in feature space, where each x_i is a variable taking values from $\mathbb{D}_i$. The set of variables associated with features is $X = \{x_1, \ldots, x_m\}$. Moreover, the notation $\mathbf{v} = (v_1, \ldots, v_m)$ represents a specific point in feature space, where each v_i is a constant representing one concrete value from $\mathbb{D}_i = \{0, 1\}$. An *instance* (or example) denotes a pair $(\mathbf{v}, c)$, where $\mathbf{v} \in \mathbb{F}$ and $c \in \mathcal{K}$. (We also use the term *instance* to refer to $\mathbf{v}$, leaving c implicit.) An ML classifier $\mathbb{C}$ is characterized by a *classification function* κ that maps feature space $\mathbb{F}$ into the set of classes $\mathcal{K}$, i.e. $\kappa : \mathbb{F} \to \mathcal{K}$. (It is assumed throughout that κ is not constant, i.e. there are at least two points $\mathbf{v}_1$ and $\mathbf{v}_2$ in feature space, where $\kappa(\mathbf{v}_1) \neq \kappa(\mathbf{v}_2)$.)

We now define formal explanations. Prime implicant (PI) explanations [52] denote a minimal set of literals (relating a feature value x_i and a constant $v_i \in \mathbb{D}_i$) that are sufficient for the prediction[2]. Formally, given $\mathbf{v} = (v_1, \ldots, v_m) \in \mathbb{F}$ with $\kappa(\mathbf{v}) = c$, a *weak* (or non-minimal) *abductive explanation* (weak AXp) is any subset $\mathcal{X} \subseteq \mathcal{F}$ such that,

$$\forall(\mathbf{x} \in \mathbb{F}). \left[\bigwedge\nolimits_{i \in \mathcal{X}} (x_i = v_i)\right] \rightarrow (\kappa(\mathbf{x}) = c) \tag{1}$$

Any subset-minimal weak AXp is referred to as an AXp. AXps can be viewed as answering a 'Why?' question, i.e. why is some prediction made given some point in feature space. A different view of explanations is a contrastive explanation [39], which answers a 'Why Not?' question, i.e. which features can be changed to change the prediction. A formal definition of *contrastive explanation* (CXp) is proposed in recent work [26]. Given $\mathbf{v} = (v_1, \ldots, v_m) \in \mathbb{F}$ with $\kappa(\mathbf{v}) = c$, a *weak* (or non-minimal) CXp is any subset $\mathcal{Y} \subseteq \mathcal{F}$ such that,

$$\exists(\mathbf{x} \in \mathbb{F}). \bigwedge\nolimits_{j \in \mathcal{F} \setminus \mathcal{Y}} (x_j = v_j) \land (\kappa(\mathbf{x}) \neq c) \tag{2}$$

Any subset-minimal weak CXp is referred to as a CXp. Building on the results of R. Reiter in model-based diagnosis [46], [26] proves a minimal hitting set (MHS) duality relation between AXps and CXps, i.e. AXps are MHSes of CXps and vice-versa.

Knowledge compilation map. Following earlier work [13, 16, 24], we define negated normal form (NNF), decomposable NNF (DNNF), deterministic DNNF (d-DNNF), decision DNNF (dec-DNNF), and also smooth d-DNNF (sd-DNNF).

▶ **Definition 1** (KC languages [16]). [3] *The following KC languages are studied in the paper:*

- *The language* negated normal form *(*NNF*) is the set of all directed acyclic graphs, where each leaf node is labeled with either $\top$, $\bot$, x_i or $\neg x_i$, for $x_i \in X$. Each internal node is labeled with either $\land$ (or* AND*) or $\lor$ (or* OR*).*
- *The language* decomposable NNF *(*DNNF*) is the set of all NNFs, where for every node labeled with $\land$, $\alpha = \alpha_1 \land \cdots \land \alpha_k$, no variables are shared between the conjuncts α_j.*

[2] PI-explanations are related with abduction, and so are also referred to as abductive explanations (AXp) [27]. More recently, PI-explanations have been studied from a knowledge compilation perspective [4, 3].

[3] We introduce KC languages that have been studied in earlier works [16, 20, 14, 43, 31]. For the sake of brevity, we define only the KC languages that are analyzed in greater detail in the paper. For the additional KC languages that are mentioned in the paper, the following references give standard definitions: OBDD [16], PI [16], IP [16], renH-C [20], AFF [20], SDD [14], dFSD [43], and EADT [31].

- *A* d-DNNF *is a* DNNF*, where for every node labeled with* $\vee$*,* $\beta = \beta_1 \vee \cdots \vee \beta_k$*, each pair* β_p, β_q*, with* $p \neq q$*, is inconsistent, i.e.* $\beta_p \wedge \beta_q \models \bot$*.*
- *An* sd-DNNF *is a* d-DNNF*, where for every node labeled with* $\vee$*,* $\beta = \beta_1 \vee \cdots \vee \beta_k$*, each pair* β_p, β_q *is defined on the same set of variables.*

The focus of this paper is d-DNNF, but for simplicity of algorithms, sd-DNNF is often considered [13]. Moreover, the definition of SDD is assumed [14, 9] (which is briefly overview in Subsection 3.4).

Throughout the paper, a term ρ denotes a conjunction of literals. A term ρ is consistent ($\rho \not\models \bot$) if the term is satisfied in at least one point in feature space.

For the purposes of this paper, we will consider exclusively the queries **CO** and **VA**, and the transformation **CD**, which we define next. Let **L** denote a subset of NNF. Hence, we have the following standard definitions [16].

▶ **Definition 2** (Conditioning [16])**.** [4] *Let* Δ *represent a propositional formula and let* ρ *denote a consistent term. The* conditioning *of* Δ *on* ρ*, denoted* $\Delta|_\rho$ *is the formula obtained by replacing each variable* x_i *by* $\top$ *(resp.* $\bot$*) if* x_i *(resp.* $\neg x_i$*) is a positive (resp. negative) literal of* ρ*.*

▶ **Definition 3** (Queries & transformations [16])**.** *The following queries and transformations are used throughout with respect to a KC language* **L***:*

- **L** *satisfies the consistency (validity) query* **CO** *(***VA***) iff there exists a polynomial-time algorithm that maps every formula* Δ *from* **L** *to 1 if* Δ *is consistent (valid), and to 0 otherwise.*
- **L** *satisfies the conditioning transformation* **CD** *iff there exists a polynomial-time algorithm that maps every formula* Δ *from* **L** *and every consistent term* ρ *into a formula that is logically equivalent to* $\Delta|_\rho$*.*

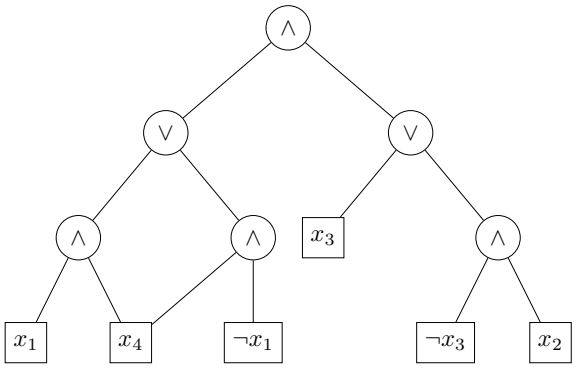
There are additional queries and transformations of interest [16], but these are beyond the goals of this paper. d-DNNF has been studied in detail from the perspective of the knowledge compilation (KC) map [16]. Hence, it is known that d-DNNF satisfies the queries **CO**, **VA**, **CE**, **IM**, **CT**, **ME**, and the transformation **CD**.

▶ **Example 4.** Figure 1 shows the running example used throughout the paper. $\mathcal{F} = \{1, 2, 3, 4\}$, $X = \{x_1, x_2, x_3, x_4\}$, and $\kappa(x_1, x_2, x_3, x_4) = ((x_1 \wedge x_4) \vee (\neg x_1 \wedge x_4)) \wedge (x_3 \vee (\neg x_3 \wedge x_2))$. Moreover, the paper considers the concrete instance $(\mathbf{v}, c) = ((0, 0, 0, 0), 0)$.

Canonical KC languages. Some widely used KC languages are canonical, i.e. equivalent functions have the same representation. Concrete examples include[5] reduced ordered decision diagrams (OBDDs) [10, 16], reduced ordered multi-valued decision diagrams MDDs [54, 7], but also sentential decision diagrams SDDs [14]. (Although we use the acronyms that are used in the literature, all these canonical representations involve some fixed order of the variables, and the resulting representation is reduced.) As shown later, for the purposes of this paper, canonicity can play a crucial role in reducing the complexity of explanation algorithms.

[4] We introduce the KC queries and transformations that are relevant for the results in the paper. There are additional queries (e.g. **CE**, **IM**, **EQ**, **SE**, **CT**, **ME**) and transformations (e.g. **FO**, **SFO**, $\wedge$**C**, $\wedge$**BC**, $\vee$**C**, $\vee$**BC**, $\neg$**C**), but are omitted for the sake of brevity. The interested reader is referred for example to [16].

[5] The paper briefly covers examples of canonical KC languages but, for the sake of brevity, does not define them. Definitions can be found in the references provided.



(a) d-DNNF $\mathcal{C}$ for $\kappa(x_1, x_2, x_3, x_4) = ((x_1 \land x_4) \lor (\neg x_1 \land x_4)) \land (x_3 \lor (\neg x_3 \land x_2))$.

x_1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$\kappa(x_1, x_2, x_3, x_4)$	0	0	0	1	0	1	0	1	0	0	0	1	0	1	0	1

(b) Truth table for d-DNNF $\mathcal{C}$. Throughout the paper, the instance considered is $\mathbf{v} = (0, 0, 0, 0)$, with prediction $c = 0$.

Figure 1 Running example (adapted from [17]).

Related Work. PI-explanations have been studied in a growing number of works [52, 53, 27, 28, 15, 4, 6, 26, 57, 37, 3]. Some of these earlier works studied PI-explanations for KC languages [52, 53, 15, 4, 6, 3]. However, results on the efficient computation of explanations for well-known KC languages are scarce. For example, [52, 53, 15] propose compilation algorithms (which are worst-case exponential) to generate the PI-explanations from OBDDs. Concretely, a classifier is compiled into an OBDD, which is then compiled into an OBDD representing the PI-explanations of the original classifier. Furthermore, [4] proves that if a KC language satisfies **CD**, **FO**, and **IM**, then one PI-explanation can be computed in polynomial time. Unfortunately, a large number of KC languages of interest do *not* simultaneously satisfy **CD**, **FO**, and **IM**. This is the case for example with OBDD, SDD, d-DNNF, among others. Moreover, [4] proves that there are polynomial time algorithms for d-DNNF for a number of XAI-relevant queries, with the exception of DPI (deriving a prime implicant explanation). Finally, Barceló et al. [6] focus on smallest PI-explanations, and prove a number of NP-hardness results.

Knowledge compilation (KC) languages also find a growing range of applications in constraint programming. Concrete examples include the compilation of constraints into Multi-Valued Decision Diagrams [22, 17] (and their use in the context of multi-objective optimization [7], among a number of other use cases) or d-DNNFs [17], but also for restoring consistency and computing explanations of dynamic CSPs [1], among others. Although explanations for classifiers find a growing interest in ML and related fields, explanations of KC languages can also find a wider range of applications, including reasoning about compiled constraints. Moreover, even though recent years have witnessed a growing interest in finding explanations of machine learning (ML) models [35, 23, 59, 41], explanations have been studied from different perspectives and in different branches of AI at least since the

Algorithm 1 Finding one AXp

Input: Classifier κ, instance $\mathbf{v}$
Output: AXp $\mathcal{S}$

1: **procedure** oneAXp($\kappa, \mathbf{v}$)
2: $\quad\mathcal{S} \leftarrow \{1, \ldots, m\}$
3: $\quad$**for** $i \in \{1, \ldots, m\}$ **do**
4: $\quad\quad$**if** isWeakAXp($\mathcal{S} \setminus \{i\}, \kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}}) = c$) **then**
5: $\quad\quad\quad\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$
6: $\quad$**return** $\mathcal{S}$

80s [51, 19, 45], including more recently in constraint programming [1, 8, 21]. The use of NP oracles for computing explanations has also been investigated in recent years [27, 37], where the NP oracle can represent a CP/SMT/MILP reasoner. (With a mild abuse of notation, when we refer to an NP oracle it is assumed that for the accepted instances, a *witness* will be returned by the oracle.)

3 Explanations for d-DNNF & Related Languages

As will be shown in this section, there is a tight connection between the definitions of AXp and CXp (see (1) and (2)) and the queries **VA**, **CO** and the transformation **CD**. Indeed, for (1) and (2), **CD** can serve to impose that the values of some features (i, represented by variable x_i) are fixed to some value v_i. In addition, **VA** (resp. **CO**) is used to decide (1), after conditioning, when $c = 1$ (resp. $c = 0$). Similarly, **VA** (resp. **CO**) is used to decide (2), again after conditioning, when $c = 1$ (resp. $c = 0$). Thus, for languages respecting the (poly-time) queries **VA** and **CO** and the (poly-time) transformation **CD**, one can compute one AXp and one CXp in polynomial time. The next sections formalize this intuition. Furthermore, even though our focus is the d-DNNF KC language, we also show that results in this section apply to any KC language respecting the queries **CO**, **VA** and the transformation **CD**.

3.1 Finding one AXp

This section details an algorithm to find one AXp. We identify any $\mathcal{S} \subseteq \{1, \ldots, m\}$ with its corresponding bit-vector $\mathbf{s} = (s_1, \ldots, s_m)$ where $s_i = 1 \Leftrightarrow i \in \mathcal{S}$. Given vectors $\mathbf{x}, \mathbf{v}, \mathbf{s}$, we can construct the vector $\mathbf{x}^{\mathbf{s},\mathbf{v}}$ (in which $\mathbf{s}$ is a selector between the two vectors $\mathbf{x}$ and $\mathbf{v}$) such that

$$x_i^{\mathbf{s},\mathbf{v}} = (x_i \land \overline{s_i}) \lor (v_i \land s_i) \tag{3}$$

To find an AXp, i.e. a subset-minimal weak AXp, Algorithm 1 is used. (Algorithm 1 is a general greedy algorithm that is well-known and used in a wide range of settings, e.g. minimal unsatisfiable core extraction in CSPs [11, 5]; to the best of our knowledge, its use in finding AXps (and also CXps) of KC languages is novel. An alternative would be to use the QuickXplain algorithm [30].)

Considering $\mathbf{s}$ and $\mathbf{v}$ as constants, when $c = 1$, $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}})$ is valid iff S is a weak AXp of $\kappa(\mathbf{v}) = c$. Furthermore, when $c = 0$, $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}})$ is inconsistent iff S is a weak AXp of $\kappa(\mathbf{v}) = c$. We therefore have the following proposition.

▶ **Proposition 5.** *For a classifier implemented with some KC language* **L***, finding one AXp is polynomial-time provided the following three operations can be performed in polynomial time:*

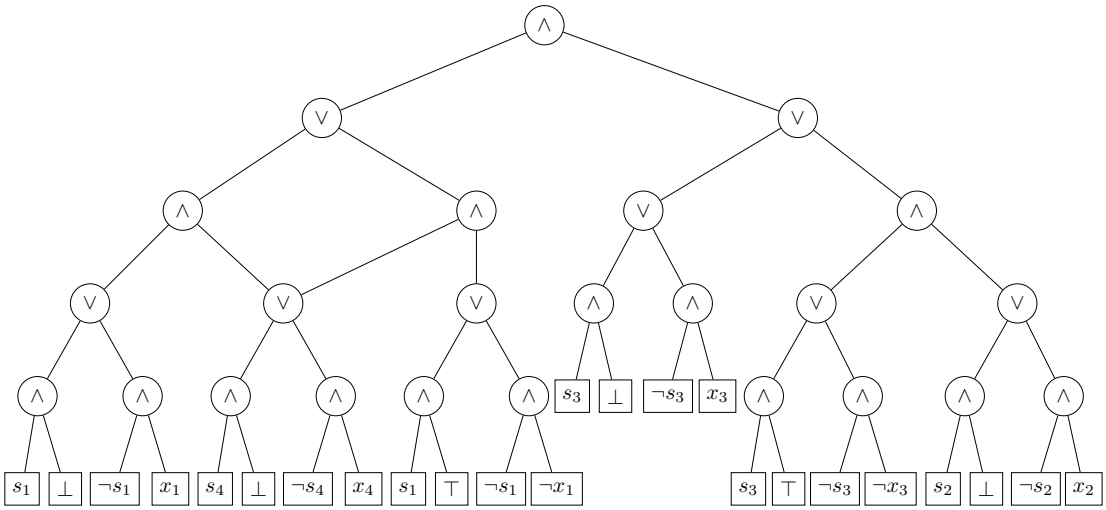


Figure 2 Modified d-DNNF, computing $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}})$ for the instance $\mathbf{v} = (0,0,0,0)$. For any pick of elements to include in the weak AXp, $\mathbf{s}$ represents constant values.

1. *construction of* $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}})$ *from* κ, $\mathbf{s}$ *and* $\mathbf{v}$.
2. *testing validity of* $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}})$.
3. *testing consistency of* $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}})$.

▶ **Corollary 6.** *Finding one AXp of a decision taken by a d-DNNF is polynomial-time.*

Proof. It is sufficient to show that d-DNNF's satisfy the conditions of Proposition 5. It is well known that testing consistency and validity d-DNNF's can be achieved in polynomial time [16]. To transform a d-DNNF calculating $\kappa(\mathbf{v})$ into a d-DNNF calculating $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}})$, we need to replace each leaf labelled x_i by a leaf labelled $(x_i \land \overline{s_i}) \lor (v_i \land s_i)$ and each leaf labelled $\overline{x_i}$ by a leaf labelled $(\overline{x_i} \land \overline{s_i}) \lor (\overline{v_i} \land s_i)$. Note that $\mathbf{s}$ and $\mathbf{v}$ are constants during this construction. Thus, we simplify these formulas to obtain either a literal or a constant according to the different cases:

- $s_i = 0$: label $(x_i \land \overline{s_i}) \lor (v_i \land s_i)$ is x_i and label $(\overline{x_i} \land \overline{s_i}) \lor (\overline{v_i} \land s_i)$ is $\overline{x_i}$. In other words, the label of the leaf node is unchanged.
- $s_i = 1$: label $(x_i \land \overline{s_i}) \lor (v_i \land s_i)$ is the (constant) value of v_i and label $(\overline{x_i} \land \overline{s_i}) \lor (\overline{v_i} \land s_i)$ is the (constant) value of $\overline{v_i}$.

Indeed, this is just conditioning (**CD**, i.e. fixing a subset of the variables x_i, given by the set S, to v_i) and it is well known that **CD** is a polytime operation on d-DNNFs [16]. ◀

▶ **Corollary 7.** *Finding one AXp of a decision taken by a classifier is polynomial-time if the classifier is given in one of the following languages: cd-PDAG [56], SDD [14], OBDD [16], PI [16], IP [16], renH-C [20], AFF [20], dFSD [43], and EADT [31].*

Proof. It suffices to show that the languages listed above satisfy the conditions of Proposition 5. According to [16], the queries **CO** and **VA** together with the transformation **CD** can all be performed in polynomial time for any of the languages listed above. This is exactly what we need to satisfy the three conditions of Proposition 5. ◀

▶ **Example 8.** The operation of the algorithm is illustrated for the d-DNNF from Example 4. By applying (3), the d-DNNF of Figure 2 is obtained. The execution of the algorithm is

i	$\mathbf{s}$	$\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}})$	Justification	Decision
1	$(0,1,1,1)$	0	$s_4 = 1$: left branch takes value 0, and so $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}}) = 0$	Drop 1
2	$(0,0,1,1)$	0	$s_4 = 1$: left branch takes value 0, and so $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}}) = 0$	Drop 2
3	$(0,0,0,1)$	0	$s_4 = 1$: left branch takes value 0, and so $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}}) = 0$	Drop 3
4	$(0,0,0,0)$	1	Simply set $\mathbf{x} = (1,1,1,1)$, and so $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}}) = 1$	Keep 4

Table 1 Example of finding one AXp

Algorithm 2 Finding one CXp

Input: Classifier κ, instance $\mathbf{v}$
Output: CXp $\mathcal{S}$

1: **procedure** oneCXp($\kappa, \mathbf{v}$)
2: $\quad\mathcal{S} \leftarrow \{1,\ldots,m\}$
3: $\quad$**for** $i \in \{1,\ldots,m\}$ **do**
4: $\quad\quad$**if** isWeakCXp($\mathcal{S} \setminus \{i\}, \kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}}) = c$) **then**
5: $\quad\quad\quad\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$
6: $\quad$**return** $\mathcal{S}$

summarized in Table 1. By inspection, we can observe that the value computed by the d-DNNF will be 0 as long as $s_4 = 1$, i.e. as long as 4 is part of the weak AXp. If removed from the weak AXp, one can find an assignment to $\mathbf{x}$, which sets $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}}) = 1$. The computed AXp is $\mathcal{S} = \{4\}$.

3.2 Finding one CXp

To compute one CXp, (2) is used. In this case, we identify any $\mathcal{S} \subseteq \{1,\ldots,m\}$ with its corresponding bit-vector $\mathbf{s}$ where $s_i = 1 \Leftrightarrow i \in \mathcal{F} \setminus \mathcal{S}$. Moreover, we adapt the approach used for computing one AXp, as shown in Algorithm 2. (Observe that the main difference is the relationship between $\mathcal{S}$ and $\mathbf{s}$, and the test for a weak CXp, that uses (2) with $\mathcal{Y} = \mathcal{S}$. Also, recall from Section 2 that κ is assumed not to be constant, and so a CXp can always be computed.)

▶ **Proposition 9.** *For a classifier implemented with some KC language* $\mathbf{L}$*, finding one CXp is polynomial-time provided the operations of Proposition 5 can be performed in polynomial time.*

▶ **Corollary 10.** *Finding one CXp of a decision taken by a classifier is polynomial-time if the classifier is given in one of the following languages: d-DNNF [16], cd-PDAG [56], SDD [14], OBDD [16], PI [16], IP [16], renH-C [20], AFF [20], dFSD [43], and EADT [31].*

▶ **Example 11.** The operation of the algorithm for computing one CXp is illustrated for the modified d-DNNF shown in Figure 2 for the instance $(\mathbf{v}, c) = ((0,0,0,0), 0)$. The execution of the algorithm is summarized in Table 2.

By inspection, we can observe that the value computed by the d-DNNF can be changed to 1 as long as $s_3 = 0 \land s_4 = 0$, i.e. as long as $\{3,4\}$ are part of the weak CXp. If removed from the weak CXp, one no longer can find an assignment to $\mathbf{x}$ that sets $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}}) = 1$. Thus, the computed CXp is $\mathcal{S} = \{3,4\}$.

i	$\mathbf{s}$	$\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}})$	Justification	Decision
1	$(1,0,0,0)$	1	Pick $\mathbf{x} = (0,1,1,1)$, and so $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}}) = 1$	Drop 1
2	$(1,1,0,0)$	1	Pick $\mathbf{x} = (0,0,1,1)$, and so $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}}) = 1$	Drop 2
3	$(1,1,1,0)$	0	$s_3 = 1$: right branch takes value 0, and so $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}}) = 0$	Keep 3
4	$(1,1,0,1)$	0	$s_4 = 1$: left branch takes value 0, and so $\kappa(\mathbf{x}^{\mathbf{s},\mathbf{v}}) = 0$	Keep 4

Table 2 Example of finding one CXp

3.3 Enumerating AXps/CXps

This section proposes a MARCO-like algorithm [34] for on-demand enumeration of AXps and CXps. For that, we need to devise modified versions of Algorithm 1 and Algorithm 2, which allow for some initial set of features (i.e. a seed) to be specified. The seed is used for computing the next AXp or CXp, and it is picked such that repetition of explanations is disallowed. As argued below, the algorithm's organization ensures that computed explanations are not repeated. Moreover, since the algorithms for computing one AXp or one CXp run in polynomial time, then the enumeration algorithm is guaranteed to require exactly one call to an NP oracle for each computed explanation, in addition to procedures that run in polynomial time.

The main building blocks of the enumeration algorithm are: (1) finding one AXp given a seed (see Algorithm 3); (2) finding one CXp given a seed (see Algorithm 4); and (3) a top-level algorithm that ensures that previously computed explanations are not repeated (see Algorithm 5). The top level-algorithm invokes a SAT oracle[6] to identify the seed which will determine whether a fresh AXp or CXp will be computed in the next iteration.

Algorithm 3 shows the computation of one AXp given an initial (seed) set of features, such that any AXp that is a subset of the given initial set of features is guaranteed not to have already been computed. Moreover, Algorithm 4 shows the computation of one CXp. As argued earlier in Sections 3.1 and 3.2, the two algorithms use one transformation, specifically conditioning (**CD**, see line 3) and two queries, namely consistency and validity (**CO**/**VA**, see line 4). In the case of computing one AXp, if the prediction is $\top$, we need to check validity, i.e. for all (conditioned) assignments, the prediction is also $\top$. In contrast, if the prediction is $\bot$, then we need to check that consistency does not hold, i.e. for all (conditioned) assignments, the prediction is also $\bot$. In contrast, in the case of computing one CXp, we need to change the tests that are executed, since we seek to change the value of the prediction. It should be noted that, by changing the conditioning operation, different KC languages can be explained; this is illustrated in Subsection 3.4. Finally, Algorithm 5 shows the proposed approach for enumerating AXps and CXps, which adapts the basic MARCO algorithm for enumerating minimal unsatisfiable cores [33]. From the definitions, we can see that for any $\mathcal{S} \subseteq \mathcal{F}$, either $\mathcal{S}$ is a weak AXp or $\mathcal{F} \setminus \mathcal{S}$ is a weak CXp. Every set $\mathcal{S}$ calculated at line 6 of Algorithm 5 has the property that it is not a superset of any previously found AXp (thanks to the clauses added to $\mathcal{H}$ at line 11) and that $\mathcal{F} \setminus \mathcal{S}$ is not a superset of any previously found CXp (thanks to the clauses added at line 15).

▶ **Example 12.** Table 3 summarizes the main steps of enumerating the AXps and CXps of

[6] A SAT oracle can be viewed as a modified NP oracle, that besides accepting/rejecting an instance (in this concrete case the formula), it also returns a satisfying assignment when the instance is satisfiable.

Algorithm 3 Finding one AXp given starting seed $\mathcal{S}$

Input: Classifier κ, Seed Set $\mathcal{S}$, Instance $\mathbf{v}$, Class c, Conditioner ς_A
Output: AXp $\mathcal{S}$

procedure findAXp($\kappa, \mathcal{S}, \mathbf{v}, c, \varsigma_A$)
  for all $i \in \mathcal{S}$ **do**
    $\kappa|_{\mathbf{s},\mathbf{v}} \leftarrow \varsigma_A(\kappa, \mathcal{S} \setminus \{i\}, \mathbf{v})$
    if $[c = \top \land \mathsf{isValid}(\kappa|_{\mathbf{s},\mathbf{v}})]$ **or** $[c = \bot \land$ **not** $\mathsf{isConsistent}(\kappa|_{\mathbf{s},\mathbf{v}})]$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$
  return $\mathcal{S}$

Algorithm 4 Finding one CXp given starting seed $\mathcal{S}$

Input: Classifier κ, Seed Set $\mathcal{S}$, Instance $\mathbf{v}$, Class c, Conditioner ς_C
Output: CXp $\mathcal{S}$

procedure findCXp($\kappa, \mathcal{S}, \mathbf{v}, c, \varsigma_C$)
  for all $i \in \mathcal{S}$ **do**
    $\kappa|_{\mathbf{s},\mathbf{v}} \leftarrow \varsigma_C(\kappa, \mathcal{S} \setminus \{i\}, \mathbf{v})$
    if $[c = \top \land$ **not** $\mathsf{isValid}(\kappa|_{\mathbf{s},\mathbf{v}})]$ **or** $[c = \bot \land \mathsf{isConsistent}(\kappa|_{\mathbf{s},\mathbf{v}})]$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \setminus \{i\}$
  return $\mathcal{S}$

the running example (see Figure 1). It is easy to confirm that after four explanations are computed, $\mathcal{H}$ becomes inconsistent, and so the algorithm terminates. Also, one can confirm the hitting set duality between AXps and CXps [26].

3.4 Explanations for SDDs

As a subset of the d-DNNF language, SDDs represent a well-known KC language [14, 18, 9]. SDDs are based on a strongly deterministic decomposition [14], which is used to decompose a Boolean function into the form: $(p_1 \land s_1) \lor \cdots \lor (p_n \land s_n)$, where each p_i is called a *prime* and each s_i is called a *sub* (both primes and subs are sub-functions). Furthermore, the process of decomposition is governed by a variable tree (*vtree*) which stipulates the variable order [14]. Figure 3 shows the SDD representation of decision function κ in Figure 1a and its *vtree* in Figure 3b.

In order to exploit Algorithm 3, 4 and 5 to explain SDD classifiers, we need to implement: (i) $\mathsf{isConsistent}$, (ii) $\mathsf{isValid}$, and (iii) the conditioning of decision function κ w.r.t. $\mathbf{s}$ and $\mathbf{v}$ (i.e. $\kappa|_{\mathbf{s},\mathbf{v}}$). To compute $\kappa|_{\mathbf{s},\mathbf{v}}$, we check each s_i if $(s_i = 1)$ and we compute $\kappa|_{x_i=v_i}$ ($\kappa|_{x_i}$ if $v_i = 1$, otherwise $\kappa|_{\neg x_i}$). As SDDs satisfy **CO**, **VA** and **CD** [18], the tractability of $\mathsf{isConsistent}$, $\mathsf{isValid}$, and $\kappa|_{\mathbf{s},\mathbf{v}}$ is guaranteed.

Next, let us consider again the running example of Figure 1 and the instance $\mathbf{v} = (0, 0, 0, 0)$ (such that $\kappa(\mathbf{v}) = 0$). Figure 4 illustrates the process of computing one AXp for $\kappa(\mathbf{v})$, which corresponds to the overall flow shown in Table 1. (Note that the computation of a CXp is similar.) As SDDs in Figures 4a, 4b and 4c are inconsistent, features 1, 2 and 3 are not necessary for preserving the prediction $\kappa(\mathbf{v}) = 0$, that is they can be removed from $\mathcal{S}$. Instead, for SDD in Figure 4d, there exists a point $\mathbf{x}$ that can be classified as $\top$, so feature 4 cannot be removed from $\mathcal{S}$. Thus, we derive an AXp $\mathcal{S} = \{4\}$.

Algorithm 5 Enumeration algorithm

Input: Feature Set $\mathcal{F}$, Classifier κ, Instance $\mathbf{v}$, Class c, Conditioners ς_A, ς_C

procedure Enumerate($\mathcal{F}, \kappa, \mathbf{v}, c, \varsigma_A, \varsigma_C$)
  $\mathcal{H} \leftarrow \emptyset$ // $\mathcal{H}$ defined on set $P = \{p_1, \ldots, p_m\}$
  repeat
    $(\mathsf{outc}, \mathbf{p}) \leftarrow \mathsf{SAT}(\mathcal{H})$
    if outc $=$ **true then**
      $\mathcal{S} \leftarrow \{i \in \mathcal{F} \,|\, p_i = 1\}$
      $\kappa|_{\mathbf{s},\mathbf{v}} \leftarrow \varsigma_A(\kappa, \mathcal{S}, \mathbf{v})$
      if $[c = \top \land \mathsf{isValid}(\kappa|_{\mathbf{s},\mathbf{v}})]$ **or** $[c = \bot \land$ **not** $\mathsf{isConsistent}(\kappa|_{\mathbf{s},\mathbf{v}})]$ **then**
        $X \leftarrow \mathsf{findAXp}(\kappa, \mathcal{S}, \mathbf{v}, c, \varsigma_A)$
        $\mathsf{reportAXp}(X)$
        $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\vee_{i \in X} \neg p_i)\}$
      else
        $X \leftarrow \mathsf{findCXp}(\kappa, \mathcal{F} \setminus \mathcal{S}, \mathbf{v}, c, \varsigma_C)$
        $\mathsf{reportCXp}(X)$
        $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\vee_{i \in X} p_i)\}$
  until outc $=$ **false**

$\mathcal{H}$	SAT($\mathcal{H}$)	$\mathbf{p}$	AXp(1), CXp(0)?	$\mathcal{S}$	AXp	CXp	Block
$\emptyset$	1	$(1,1,1,1)$	1	$\{1,2,3,4\}$	$\{4\}$	—	$b_1 = (\neg p_4)$
$\{b_1\}$	1	$(1,1,1,0)$	1	$\{1,2,3\}$	$\{2,3\}$	—	$b_2 = (\neg p_2 \vee \neg p_3)$
$\{b_1, b_2\}$	1	$(1,0,1,0)$	0	$\{1,3\}$	—	$\{2,4\}$	$b_3 = (p_2 \vee p_4)$
$\{b_1, b_2, b_3\}$	1	$(1,1,0,0)$	0	$\{1,2\}$	—	$\{3,4\}$	$b_4 = (p_3 \vee p_4)$
$\{b_1, b_2, b_3, b_4\}$	0	—	—	—	—	—	—

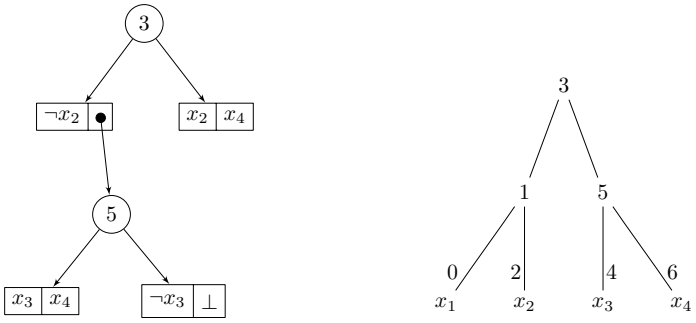
Table 3 Example of AXp/CXp enumeration, using Algorithm 5

4 Generalizations

4.1 Explanations for Generalized Decision Functions

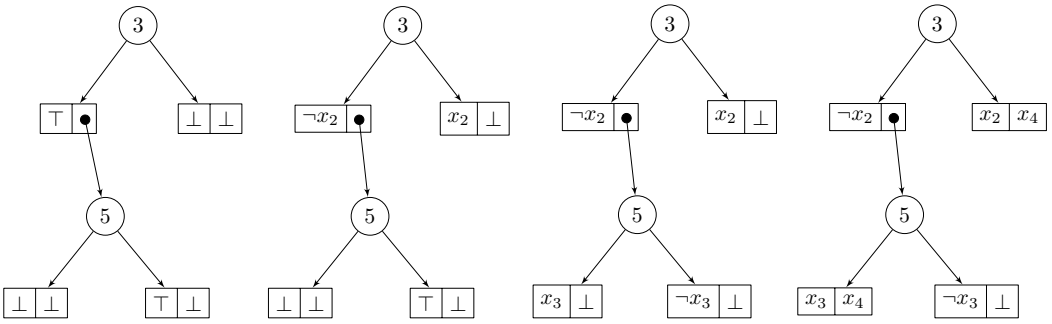
We consider the setting of multi-class classification, with $\mathcal{K} = \{c_1, \ldots, c_K\}$, where each class c_j is associated with a total function $\kappa_j : \mathbb{F} \to \{0, 1\}$, such that the class c_j is picked iff $\kappa_j(\mathbf{v}) = 1$. For example, *decision sets* [32] represent one such example of multi-class classification, where each function κ_j is represented by a DNF, and a *default rule* is used to pick some class for the points $\mathbf{v}$ in feature space for which all $\kappa_j(\mathbf{v}) = 0$. Moreover, decision sets may exhibit *overlap* [29], i.e. the existence of points $\mathbf{v}$ in feature space such that there exist $j_1 \neq j_2$ and $\kappa_{j_1}(\mathbf{v}) = \kappa_{j_2}(\mathbf{v}) = 1$. In practice, the existence of overlap can be addressed by randomly picking one of the classes for which $\kappa_j(\mathbf{v}) = 1$. Alternatively, the learning of DSes can ensure that overlap is non-existing [29].

This section considers generalized versions of DSes, by removing the restriction that each class is computed with a DNF. Hence, a *generalized decision function* (GDF) is such that each function κ_j is allowed to be an *arbitrary* boolean function. Furthermore, the following two properties of GDFs are considered:



(a) SDD representation **(b)** vtree

Figure 3 SDD for $\kappa(x_1, x_2, x_3, x_4) = ((x_1 \wedge x_4) \vee (\neg x_1 \wedge x_4)) \wedge (x_3 \vee (\neg x_3 \wedge x_2))$, given a vtree. Each circle node with outgoing edges is a *decision node* while each paired-boxes node is an *element*. The left (resp. right) box represents the *prime* (resp. *sub*). A box either contains a terminal SDD (i.e. $\top$, $\bot$ or a literal) or a link to a *decision node*. The shown *vtree* in (3b) is a binary tree, whose leaves are in a one-to-one correspondence with the domain variables of $\kappa(x_1, x_2, x_3, x_4)$. Moreover, each SDD node *respects* a unique (leaf or non-leaf) node of the *vtree*, e.g. the SDD root of (3a) respects the vtree root of (3b).



(a) $\mathbf{s} = (0, 1, 1, 1)$ **(b)** $\mathbf{s} = (0, 0, 1, 1)$ **(c)** $\mathbf{s} = (0, 0, 0, 1)$ **(d)** $\mathbf{s} = (0, 0, 0, 0)$

Figure 4 Example of computing one AXp for $\kappa(\mathbf{v} = (0, 0, 0, 0)) = 0$. Each sub-figure represents an SDD $\kappa|_{\mathbf{s},\mathbf{v}}$. (Note that, the SDDs are presented in intermediate form for better illustrating the procedure of calculating the explanation.)

.

▶ **Definition 13** (Binding GDF). *A GDF is* binding *if,*

$$\forall(\mathbf{x} \in \mathbb{F}). \bigvee_{1 \leq j \leq K} \kappa_j(\mathbf{x}) \tag{4}$$

(Thus, a binding GDF requires no default rule, since for any point $\mathbf{x}$ in feature space, there is at least one κ_j such that $\kappa_j(\mathbf{x})$ holds.)

▶ **Definition 14** (Non-overlapping GDF). *A GDF is* non-overlapping *if,*

$$\forall(\mathbf{x} \in \mathbb{F}). \bigwedge_{\substack{1 \leq j_1, j_2 \leq K \\ j_1 \neq j_2}} (\neg\kappa_{j_1}(\mathbf{x}) \vee \neg\kappa_{j_2}(\mathbf{x})) \tag{5}$$

(Thus, a binding, non-overlapping GDF computes a total multi-class classification function.)

Furthermore, we can establish conditions for a GDF to be binding and non-overlapping:

▶ **Proposition 15.** *A GDF is binding and non-overlapping iff the following formula is inconsistent:*

$$\exists(\mathbf{x} \in \mathbb{F}).\kappa_1(\mathbf{x}) + \ldots + \kappa_K(\mathbf{x}) \neq 1 \tag{6}$$

Proof. Given Definition 13 and Definition 14,

1. Clearly, there exists a point $\mathbf{v} \in \mathbb{F}$ such that $\kappa_1(\mathbf{v}) + \ldots + \kappa_K(\mathbf{v}) = 0$ iff the GDF is non-binding;
2. Clearly, there exists $\mathbf{v} \in \mathbb{F}$ such that $\kappa_1(\mathbf{v}) + \ldots + \kappa_K(\mathbf{v}) \geq 2$ iff the GDF is overlapping.

Thus, the result follows. ◀

▶ Remark 16. For a GDF where each function is represented by a boolean circuit, deciding whether a GDF is binding and non-overlapping is in coNP. In practice, checking whether a GDF is binding and non-overlapping can be decided with a call to an NP oracle.

▶ **Proposition 17.** *For a binding and non-overlapping GDF, such that each classification function is represented by a sentence of a KC language satisfying the query* **CO** *and the transformation* **CD***, then one AXp or one CXp can be computed in polynomial time. Furthermore, enumeration of AXps/CXps can be achieved with one call to an NP oracle per computed explanation.*

Proof sketch. For computing one AXp of class c_p, one can iteratively check consistency of the remaining of literals on the other functions $q \neq p$. Conditioning is used to reflect, in the classifiers, the choices made, i.e. which literals are included or not in the AXp. For a CXp a similar approach can be used. For enumeration, we can once again exploit a MARCO-like algorithm. ◀

▶ **Corollary 18.** *For a binding non-overlapping GDF, where each κ_j is represented by a* DNNF*, one AXp and one CXp can be computed in polynomial time. Furthermore, enumeration of AXps/CXps can be achieved with one call to an NP oracle per computed explanation.*

Thus, for GDFs that are both binding and non-overlapping, even if each function is represented by the fairly succinct DNNF, one can still compute AXps and CXps efficiently. Furthermore, a MARCO-like [34] can be used for enumerating AXps and CXps.

The results above can be generalized to the case of multi-valued classification, where binarization (one-hot-encoding) can serve for representing multi-valued (non-continuous) features. Alternative approaches have been investigated in recent work [4].

4.2 Total Congruent Classifiers

We can build on the conditions for GDFs to devise relaxed conditions for poly-time explainability.

▶ **Definition 19** (Total Classifier)**.** *A classification function is total if for any point $\mathbf{v} \in \mathbb{F}$, there is a prediction $\kappa(\mathbf{v}) = c$, with $c \in \mathcal{K}$.*

▶ **Definition 20** (Congruent Classifier)**.** *A classifier is* congruent *if the computational complexity of deciding the consistency of $\kappa(\mathbf{x}) = c$ is the same for any $c \in \mathcal{K}$.*

Similarly, we can define a total congruent KR language. For a total congruent KR language, the query **CO** is satisfied iff deciding $\kappa(\mathbf{v}) = c$ is in polynomial time for any $c \in \mathcal{K}$. Given the above, the same argument used for GDFs, can be used to prove that,

▶ **Proposition 21.** *For a total congruent KR language, which satisfies the operations of* **CO** *and* **CD***, one AXp and one CXp can be computed in polynomial time.*

Dataset	(#F	#S)	Model			XPs	AXp			CXp			d-DNNF		SDD	
			%A	#ND	#NS	avg	M	avg	%L	M	avg	%L	M	avg	M	avg
corral	(6	160)	100	35	12	4	4	2	34	4	2	22	0.004	0.001	0.001	0.000
db-bodies	(4702	64)	100	22	21	4	3	2	1	4	3	1	0.004	0.002	0.001	0.000
db-bodies-stemmed	(3721	64)	84.6	14	15	4	2	1	1	4	2	1	0.003	0.002	0.001	0.000
db-subjects	(242	64)	84.6	45	28	6	4	2	2	6	4	1	0.005	0.002	0.004	0.001
db-subjects-stemmed	(229	64)	92.3	54	31	7	4	2	2	7	5	1	0.006	0.003	0.004	0.001
mofn_3_7_10	(10	1324)	97.7	107	34	11	28	4	32	28	6	24	0.072	0.011	0.008	0.001
mux6	(6	128)	100	62	22	5	4	2	51	4	3	24	0.009	0.003	0.002	0.001
parity5+5	(10	1124)	85.7	484	96	9	12	2	66	19	7	14	0.193	0.038	0.009	0.002
spect	(22	267)	85.1	108	55	14	36	8	22	13	6	10	0.105	0.030	0.016	0.005
threeOf9	(9	512)	96.1	76	37	7	15	3	38	14	4	19	0.023	0.006	0.005	0.001
xd6	(9	973)	97.9	80	36	8	25	4	36	22	4	20	0.035	0.009	0.007	0.001

Table 4 Listing all AXps CXps for d-DNNFs and SDDs. Columns **#F** and **#S** report, resp., the number of features and the number of tested samples (instances), in the dataset. Sub-Column **%A** reports the (test) accuracy of the model and **#ND** (resp. **#NS**) shows the total number of nodes in the compiled d-DNNF (resp. SDD). Column **XPs** reports the average number of total explanations (AXp's and CXp's). Sub-columns **M** and **avg** of column **AXp** (resp., **CXp**) show, resp., the maximum and average number of explanations. The average length of an explanation (AXp/CXp) is given as **%L**. Sub-columns **M** and **avg** of column **d-DNNF** (resp. SDD) report, resp., maximal and average runtime (in seconds) to list all the explanations for all tested instances.

5 Experimental Results

In this section, we present the experiments carried out to assess the practical effectiveness of the proposed approach. The assessment is performed on the computation of AXps and CXps for d-DNNFs and SDDs. The experiments consider a selection of 11 binary datasets that are publicly available and originate from the Penn Machine Learning Benchmarks [44] and the openML repository [55]. To learn d-DNNFs (resp. SDDs), we first train Read-Once Decision Tree (RODT) models on the given binary datasets and then compile the obtained RODTs into d-DNNFs (resp. SDDs). (A RODT is a *free* BDD (FBDD) whose underlying graph is a tree [6, 58], where FBDD is defined as a BDD that satisfies the *read-once property*: each variable is encountered at most once on each path from the root to a leaf node.) The compilation of RODTs to d-DNNFs can be easily done by direct mapping, since RODT is a special case of FBDDs, and FBDDs is a subset of d-DNNFs [16] To compile SDDs, we use the PySDD package[7], which is implemented in Python and Cython. (Note that, we tuned PySDD to use *dynamic minimization* [12] during the construction in order to reduce the size of the SDDs.) The PySAT package [25] is used to instrument incremental SAT oracle calls in AXp/CXp enumeration. Lastly, The experiments are performed on a MacBook Pro with a 6-Core Intel Core i7 2.6 GHz processor with 16 GByte RAM, running macOS Big Sur.

PySDD wraps the famous SDD package[8] which offers canonical SDDs[9]. Employing canonical SDDs allows consistency and validity checking to be done in constant time (If the canonical SDD is inconsistent (resp. valid) then it is a single node labeled with $\bot$ (resp. $\top$) [14]), so in practice may improve the efficiency of explaining SDD classifiers.

Table 4 summarizes the obtained results of explaining d-DNNFs and SDDs. (Note that, for each dataset, the compiled d-DNNF and SDD represent the same decision function of the learned RODT. Hence, the computed explanations are the same as well.) Performance-wise, the maximum running time to enumerate all AXps/CXps is less than 0.2 sec for all tested

[7] https://github.com/wannesm/PySDD
[8] http://reasoning.cs.ucla.edu/sdd/
[9] Since PySDD offers canonical SDDs, the **CD** transformation is not implemented in worst-case polynomial time [18]. However, in practice, this was never an issue in our experiments.

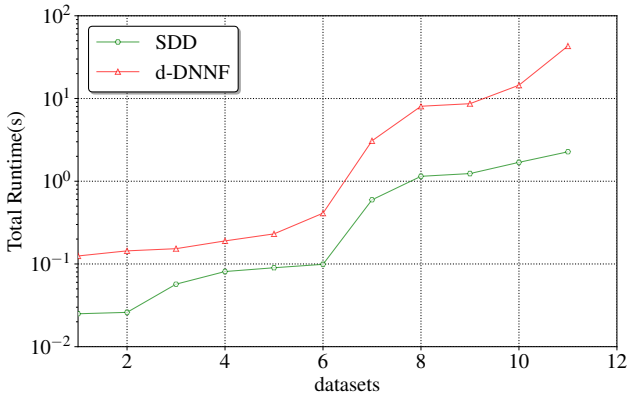


Figure 5 Comparison of total runtime (in seconds) spent to explain all instances of each dataset for d-DNNFs vs. SDDs.

d-DNNFs, and is less than 0.02 sec for all tested SDDs. On average, it takes at most 0.038 sec for enumerating all the explanations (AXps/CXps) of d-DNNFs; for SDDs, it takes a few milliseconds to enumerate all the explanations (AXps/CXps). Thus the overall cost of the SAT oracle calls performed by the enumeration algorithm is negligible. Hence, it is plain that instrumenting SAT oracle calls does not constitute a bottleneck to listing effectively all the AXps/CXps of the d-DNNFs and SDDs. Apart from the runtime, one observation is that the total number of AXps and CXps per instance is relatively small. Moreover, if compared with the total number of features, the average length of an explanation (AXp or CXp) is also relatively small.

We compared the raw performance of explaining d-DNNFs and SDDs. Figure 5 depicts a cactus plot showing the total runtime spent on AXp-and-CXp enumeration for all instances of each dataset. As can be seen, both d-DNNF and SDD explanation procedures are able to finish successful enumeration of AXps/CXps for all instances of the datasets in a few seconds. Unsurprisingly, the runtimes in case of SDDs tend to be overall better than those for d-DNNFs. Indeed, explaining SDDs is on average 6 times faster than explaining d-DNNFs. One factor contributing to this performance difference is that in practice in case of SDDs consistency and validity checking can be done in constant time.

To conclude, the results shown above, for the concrete case of classifiers represented in the d-DNNF and SDD languages, support the paper's theoretical claims from a practical side that, if the underlying KC language implements polynomial-time **CO** and **VA** queries as well as the **CD** transformation, then (i) the polynomial-time computation of one AXp/CXp in practice takes a negligible amount of time, which together with (ii) making a single SAT oracle call per explanation makes (iii) the enumeration of (some/all) XPs (AXps and CXps) highly efficient in practice.

6 Conclusions

This paper proves that for any classifier that can be represented with a d-DNNF, both one AXp and on CXp can be computed in polynomial time on the size of the d-DNNF. Furthermore, the paper shows that enumeration of AXps and CXps can be implemented with one NP oracle call per explanation. The experimental evidence confirms that for small numbers of explanations, the cost of enumeration is negligible. In addition, the paper proposes conditions for generalized decision functions to be explained in polynomial

time. Concretely, the paper develops conditions which allow generalized decision functions represented with DNNFs to be explainable in polynomial time. Finally, the paper proposes general conditions for a classifier to be explained in polynomial time. The experimental results validate the scability of the polynomial time algorithms and, more importantly, the scalability of oracle-based enumeration.

References

1 Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis. Consistency restoration and explanations in dynamic CSPs application to configuration. *Artif. Intell.*, 135(1-2):199–234, 2002. doi:10.1016/S0004-3702(01)00162-X.

2 Sule Anjomshoae, Amro Najjar, Davide Calvaresi, and Kary Främling. Explainable agents and robots: Results from a systematic literature review. In *AAMAS*, pages 1078–1088, 2019.

3 Gilles Audemard, Steve Bellart, Louenas Bounia, Frédéric Koriche, Jean-Marie Lagniez, and Pierre Marquis. On the computational intelligibility of boolean classifiers. *CoRR*, abs/2104.06172, 2021. URL: https://arxiv.org/abs/2104.06172, arXiv:2104.06172.

4 Gilles Audemard, Frédéric Koriche, and Pierre Marquis. On tractable XAI queries based on compiled representations. In *KR*, pages 838–849, 2020.

5 R. R. Bakker, F. Dikker, F. Tempelman, and P. M. Wognum. Diagnosing and solving over-determined constraint satisfaction problems. In *IJCAI*, pages 276–281, 1993.

6 Pablo Barceló, Mikaël Monet, Jorge Pérez, and Bernardo Subercaseaux. Model interpretability through the lens of computational complexity. In *NeurIPS*, 2020.

7 David Bergman, André A. Ciré, Willem-Jan van Hoeve, and John N. Hooker. *Decision Diagrams for Optimization*. Springer, 2016. doi:10.1007/978-3-319-42849-9.

8 Bart Bogaerts, Emilio Gamba, Jens Claes, and Tias Guns. Step-wise explanations of constraint satisfaction problems. In *ECAI*, pages 640–647, 2020.

9 Simone Bova. SDDs are exponentially more succinct than OBDDs. In *AAAI*, pages 929–935, 2016.

10 Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986. doi:10.1109/TC.1986.1676819.

11 John W. Chinneck and Erik W. Dravnieks. Locating minimal infeasible constraint sets in linear programs. *INFORMS J. Comput.*, 3(2):157–168, 1991. doi:10.1287/ijoc.3.2.157.

12 Arthur Choi and Adnan Darwiche. Dynamic minimization of sentential decision diagrams. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 27, 2013.

13 Adnan Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *J. Appl. Non Class. Logics*, 11(1-2):11–34, 2001. doi:10.3166/jancl.11.11-34.

14 Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *IJCAI*, pages 819–826, 2011.

15 Adnan Darwiche and Auguste Hirth. On the reasons behind decisions. In *ECAI*, pages 712–720, 2020.

16 Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17:229–264, 2002. doi:10.1613/jair.989.

17 Diego de Uña, Graeme Gange, Peter Schachte, and Peter J. Stuckey. Compiling CP subproblems to MDDs and d-DNNFs. *Constraints An Int. J.*, 24(1):56–93, 2019. doi:10.1007/s10601-018-9297-2.

18 Guy Van den Broeck and Adnan Darwiche. On the role of canonicity in knowledge compilation. In *AAAI*, pages 1641–1648, 2015.

19 Marcelo A. Falappa, Gabriele Kern-Isberner, and Guillermo Ricardo Simari. Explanations, belief revision and defeasible reasoning. *Artif. Intell.*, 141(1/2):1–28, 2002. doi:10.1016/S0004-3702(02)00258-8.

20 Hélène Fargier and Pierre Marquis. Extending the knowledge compilation map: Krom, horn, affine and beyond. In *AAAI*, pages 442–447, 2008.

21 Emilio Gamba, Bart Bogaerts, and Tias Guns. Efficiently explaining CSPs with unsatisfiable subset optimization. In *IJCAI*, 2021. In press.

22 Rebecca Gentzel, Laurent Michel, and Willem Jan van Hoeve. HADDOCK: A language and architecture for decision diagram compilation. In *CP*, pages 531–547, 2020. doi:10.1007/978-3-030-58475-7_31.

23 Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5):93:1–93:42, 2019. doi:10.1145/3236009.

24 Jinbo Huang and Adnan Darwiche. The language of search. *J. Artif. Intell. Res.*, 29:191–219, 2007. doi:10.1613/jair.2097.

25 Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018.

26 Alexey Ignatiev, Nina Narodytska, Nicholas Asher, and Joao Marques-Silva. From contrastive to abductive explanations and back again. In *AI*IA*, 2020. (Preliminary version available from https://arxiv.org/abs/2012.11067.).

27 Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. Abduction-based explanations for machine learning models. In *AAAI*, pages 1511–1519, 2019.

28 Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. On relating explanations and adversarial examples. In *NeurIPS*, pages 15857–15867, 2019.

29 Alexey Ignatiev, Filipe Pereira, Nina Narodytska, and João Marques-Silva. A SAT-based approach to learn explainable decision sets. In *IJCAR*, pages 627–645, 2018.

30 Ulrich Junker. QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In *AAAI*, pages 167–172, 2004.

31 Frédéric Koriche, Jean-Marie Lagniez, Pierre Marquis, and Samuel Thomas. Knowledge compilation for model counting: Affine decision trees. In *IJCAI*, pages 947–953, 2013.

32 Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *KDD*, pages 1675–1684, 2016.

33 Mark H. Liffiton and Ammar Malik. Enumerating infeasibility: Finding multiple MUSes quickly. In *CPAIOR*, pages 160–175, 2013.

34 Mark H. Liffiton, Alessandro Previti, Ammar Malik, and João Marques-Silva. Fast, flexible MUS enumeration. *Constraints An Int. J.*, 21(2):223–250, 2016.

35 Zachary C. Lipton. The mythos of model interpretability. *Commun. ACM*, 61(10):36–43, 2018. doi:10.1145/3233231.

36 Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *NeurIPS*, pages 4765–4774, 2017.

37 Emanuele La Malfa, Agnieszka Zbrzezny, Rhiannon Michelmore, Nicola Paoletti, and Marta Kwiatkowska. On guaranteed optimal robust explanations for NLP models. In *IJCAI*, 2021. In press.

38 Tim Miller. "But why?" understanding explainable artificial intelligence. *ACM Crossroads*, 25(3):20–25, 2019. doi:10.1145/3313107.

39 Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.*, 267:1–38, 2019. doi:10.1016/j.artint.2018.07.007.

40 Brent D. Mittelstadt, Chris Russell, and Sandra Wachter. Explaining explanations in AI. In *FAT*, pages 279–288, 2019. doi:10.1145/3287560.3287574.

41 Don Monroe. Deceiving AI. *Commun. ACM*, 64, 2021. URL: https://doi.org/10.1145/3453650.

42 Shane T. Mueller, Robert R. Hoffman, William J. Clancey, Abigail Emrey, and Gary Klein. Explanation in human-AI systems: A literature meta-review, synopsis of key ideas and publications, and bibliography for explainable AI. *CoRR*, abs/1902.01876, 2019. arXiv:1902.01876.

43 Alexandre Niveau, Hélène Fargier, and Cédric Pralet. Representing CSPs with set-labeled diagrams: A compilation map. In *GKR*, pages 137–171, 2011.

44 Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1):36, 2017.

45 Ramón Pino Pérez and Carlos Uzcátegui. Preferences and explanations. *Artif. Intell.*, 149(1):1–30, 2003. doi:10.1016/S0004-3702(03)00042-0.

46 Raymond Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.

47 Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *KDD*, pages 1135–1144, 2016.

48 Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI*, pages 1527–1535, 2018.

49 Wojciech Samek, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus-Robert Müller, editors. *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, 2019. doi:10.1007/978-3-030-28954-6.

50 Wojciech Samek and Klaus-Robert Müller. Towards explainable artificial intelligence. In Samek et al. [49], pages 5–22. doi:10.1007/978-3-030-28954-6_1.

51 Murray Shanahan. Prediction is deduction but explanation is abduction. In *IJCAI*, pages 1055–1060, 1989.

52 Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining bayesian network classifiers. In *IJCAI*, pages 5103–5111, 2018.

53 Andy Shih, Arthur Choi, and Adnan Darwiche. Compiling bayesian network classifiers into decision graphs. In *AAAI*, pages 7966–7974, 2019.

54 Arvind Srinivasan, Timothy Kam, Sharad Malik, and Robert K. Brayton. Algorithms for discrete function manipulation. In *ICCAD*, pages 92–95, 1990.

55 Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

56 Michael Wachter and Rolf Haenni. Propositional DAGs: A new graph-based language for representing boolean functions. In *KR*, pages 277–285, 2006.

57 Stephan Wäldchen, Jan MacDonald, Sascha Hauch, and Gitta Kutyniok. The computational complexity of understanding binary classifier decisions. *J. Artif. Intell. Res.*, 70:351–387, 2021. doi:10.1613/jair.1.12359.

58 Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000. URL: http://ls2-www.cs.uni-dortmund.de/monographs/bdd/.

59 Daniel S. Weld and Gagan Bansal. The challenge of crafting intelligible intelligence. *Commun. ACM*, 62(6):70–79, 2019. doi:10.1145/3282486.

60 Feiyu Xu, Hans Uszkoreit, Yangzhou Du, Wei Fan, Dongyan Zhao, and Jun Zhu. Explainable AI: A brief survey on history, research areas, approaches and challenges. In *NLPCC*, pages 563–574, 2019. doi:10.1007/978-3-030-32236-6_51.