Certified Logic-Based Explainable AI

Aurélie Hurault, Joao Marques-Silva

Certified Logic-Based Explainable AI – The Case of Monotonic Classifiers

Aurélie Hurault[1][0000−0002−3266−6080] and
Joao Marques-Silva[2][0000−0002−6632−3086]

[1] IRIT, Université de Toulouse, Toulouse, France aurelie.hurault@enseeiht.fr
[2] IRIT, CNRS, Toulouse, France joao.marques-silva@irit.fr

Abstract. The continued advances in artificial intelligence (AI), including those in machine learning (ML), raise concerns regarding their deployment in high-risk and safety-critical domains. Motivated by these concerns, there have been calls for the verification of systems of AI, including their explanation. Nevertheless, tools for the verification of systems of AI are complex, and so error-prone. This paper describes one initial effort towards the certification of logic-based explainability algorithms, focusing on monotonic classifiers. Concretely, the paper starts by using the proof assistant Coq to prove the correctness of recently proposed algorithms for explaining monotonic classifiers. Then, the paper proves that the algorithms devised for monotonic classifiers can be applied to the larger family of *stable* classifiers. Finally, confidence code, extracted from the proofs of correctness, is used for computing explanations that are guaranteed to be correct. The experimental results included in the paper show the scalability of the proposed approach for certifying explanations.

Keywords: Formal Explainability · Certification.

1 Introduction

The ongoing advances in Artificial Intelligence (AI), including in Machine Learning (ML), raise concerns about whether human decision makers can trust the decisions made by systems of AI/ML, and even whether they are able to fathom them. Aiming to address these concerns, the field of eXplainable AI (XAI) [6–8] has witnessed massive interest [5]. Explainability has also been proposed as a core component of efforts for the verification of ML models [24].

Unfortunately, most work on XAI offers no guarantees of rigor. Model-agnostic XAI approaches [14, 22, 23] represent one such example. As a result of the lack of guarantees of rigor, there is by now comprehensive evidence [9, 10, 13] that confirms the lack of rigor of non-formal XAI approaches. These results are troublesome, especially in application domains where rigor is paramount. To address the limitations of non-formal XAI, there has been work on formal XAI (FXAI) [16, 18]. Formal explanations are logically defined and model-based, and so guarantee the correctness of computed explanations, as long as (i) the representation of the ML model is adequate; and (ii) the implemented algorithms

are correct. Unfortunately, algorithms can exhibit bugs, as can their implementations. Hence, besides the need to explain and/or verify AI/ML models, and in settings that are deemed of high-risk or that are safety-critical, the certification of computed explanations is bound to become a required step.

This paper represents a first step in the direction of certifying the computation of explanations. Concretely, we use the Coq proof assistant to prove the correctness of recently proposed explanation algorithms for monotonic classifiers. The insights from the proof of correctness also serve to generalize the algorithms proposed for monotonic classifiers to a more general class of stable classifiers. Finally, the proofs of correctness are used to generate confidence code, which can be used for computing explanations that are guaranteed to be correct.

The paper is organized as follows. Section 2 introduces the notation and definitions used throughout the paper. Section 3 briefly overviews the computation of explanations in the case of monotonic classifiers [17]. Section 4 details the approach for the proofs of correctness and demonstrates that the results can be applied to a broader category of classifiers we named *stable*. Section 5 provides evidence to the scalability of certified explainers for *stable* classifiers. Finally, Section 6 concludes the paper.

2 Preliminaries

We followed the notation and definitions used in earlier work [17].

Classification problems. A classification problem is defined on a set of features $\mathcal{F} = \{1, \ldots, N\}$ and a set of classes $\mathcal{K} = \{c_1, c_2, \ldots, c_M\}$. Each feature $i \in \mathcal{F}$ takes values from a domain $\mathcal{D}_i$. Domains are ordinal and bounded, and each domain can be defined on boolean, integer or real values. If $x_i \in \mathcal{D}_i$, then $\lambda(i)$ and $\mu(i)$ denote respectively the smallest and largest values that x_i can take, i.e. $\lambda(i) \leq x_i \leq \mu(i)$. Feature space is defined by $\mathbb{F} = \mathcal{D}_1 \times \mathcal{D}_2 \times \ldots \times \mathcal{D}_N$. The notation $\mathbf{x} = (x_1, \ldots, x_N)$ denotes an arbitrary point in feature space, where each x_i is a variable taking values from $\mathcal{D}_i$. Moreover, the notation $\mathbf{v} = (v_1, \ldots, v_N)$ represents a specific point in feature space, where each v_i is a constant representing one concrete value from $\mathcal{D}_i$. An *instance* denotes a pair $(\mathbf{v}, c)$, where $\mathbf{v} \in \mathbb{F}$ and $c \in \mathcal{K}$. An ML classifier $\mathcal{M}$ is characterized by a non-constant *classification function* κ that maps feature space $\mathbb{F}$ into the set of classes $\mathcal{K}$, i.e. $\kappa : \mathbb{F} \to \mathcal{K}$.

Monotonic classifiers.
Given two points in feature space $\mathbf{a}$ and $\mathbf{b}$, $\mathbf{a} \leq \mathbf{b}$ if $a_i \leq b_i$, for all $i \in \{1, \ldots, N\}$. A set of classes $\mathcal{K} = \{c_1, \ldots, c_M\}$ is *ordered* if it respects a total order $\preccurlyeq$, with $c_1 \preccurlyeq c_2 \preccurlyeq \ldots \preccurlyeq c_M$. An ML classifier $\mathbb{M}$ is fully monotonic if the associated classification function is monotonic, i.e. $\mathbf{a} \leq \mathbf{b} \Rightarrow \kappa(\mathbf{a}) \preccurlyeq \kappa(\mathbf{b})$[3]. Throughout the paper, when referring to a monotonic classifier, this signifies a fully monotonic classifier. In addition, the interaction with a classifier is restricted to computing

[3] The paper adopts the classification of monotonic classifiers proposed in earlier work [4].

the value of $\kappa(\mathbf{v})$, for some point $\mathbf{v} \in \mathbb{F}$, i.e. the classifier will be viewed as a black-box.

Intuitively, for a monotonic classifier the relationship between the input features and the predicted outcome is always non-decreasing. This property allows to make strong causal inferences about the impact of the input features on the predicted outcome. For example, consider a credit scoring model that predicts the likelihood of a loan being repaid based on the borrower's credit history. A monotonic credit scoring model would always assign a higher probability of repayment for borrowers with a longer and more stable credit history, all other factors being equal. This property can be helpful in ensuring fairness and transparency in decision-making, as it allows us to identify and address potential biases in the model based on certain input features.

***Stable* classifiers.**
An ML classifier $\mathbb{M}$ is *stable* if the associated classification function is *stable*, i.e. $\forall \mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{F}, \mathbf{a} \preceq \mathbf{b} \preceq \mathbf{c} \land \kappa(\mathbf{a}) = \kappa(\mathbf{c}) \Rightarrow \kappa(\mathbf{a}) = \kappa(\mathbf{b}) = \kappa(\mathbf{c})$.
As for monotonic classifiers, an order relation is needed on each domain $\mathcal{D}_i$. However, classes do not need to be ordered. All monotonic classifiers are *stable*.

Intuitively, a *stable* classifier can be thought of as relaxing the requirement of monotonicity by not imposing a specific order on the classes. In other words, if two points in the feature space receive the same prediction from the classifier, then all points between them should also receive the same prediction.

Non-trivial classifiers.
An ML classifier $\mathbb{M}$ is nontrivial if the associated classification function is not constant, i.e. $\exists \mathbf{a}, \mathbf{b} \in \mathbb{F}, \kappa(\mathbf{a}) \neq \kappa(\mathbf{b})$.

Logic-based explainability. We now define formal explanations. For brevity, we only provide a brief introduction to logic-based explainability.

Prime implicant (PI) explanations [25] denote a minimal set of literals (relating a feature value x_i and a constant v_i from its domain $\mathcal{D}_i$) that are sufficient for the prediction. [4]. Formally, given $\mathbf{v} = (v_1, \ldots, v_N) \in \mathbb{F}$ with $\kappa(\mathbf{v}) = c$, an AXp is any minimal subset $\mathcal{X} \subseteq \mathcal{F}$ such that,

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

We associate a predicate WAXp with (1), such that any set $\mathcal{X} \subseteq \mathcal{F}$ for which $\mathsf{WAXp}(\mathcal{X})$ holds is referred to as a *weak* AXp. Thus, every AXp is a weak AXp that is also subset-minimal. AXp's 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 [20], which answers a 'Why

[4] PI-explanations can be formulated as a problem of logic-based abduction, and so are also referred to as abductive explanations (AXp) [12]. More recently, AXp's have been studied from a knowledge compilation perspective [1].

Not?' question, i.e. which features can be changed to change the prediction. A formal definition of contrastive explanation is proposed in recent work [11]. Given $\mathbf{v} = (v_1, \ldots, v_N) \in \mathbb{F}$ with $\kappa(\mathbf{v}) = c$, a CXp is any minimal subset $\mathcal{Y} \subseteq \mathcal{F}$ such that,

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

Moreover, we associate a predicate WCXp with (2), such that any set $\mathcal{Y} \subseteq \mathcal{F}$ for which $\mathsf{WCXp}(\mathcal{Y})$ holds is referred to as a *weak* CXp. Thus, every CXp is a weak CXp that is also subset-minimal. Building on the results of R. Reiter in model-based diagnosis [21], [11] proves a minimal hitting set (MHS) duality relation between AXp's and CXp's, i.e. AXp's are MHSes of CXp's and vice-versa. Furthermore, it can be shown that both predicates WAXp and WCXp are monotone. An important consequence of this observation is that one can then use efficient oracle-based algorithms for finding AXp's and/or CXp's [19]. Thus, as long as one can devise logic encodings for an ML classifier (and this is possible for most ML classifiers), then (1) and (2), and access to a suitable reasoner, offer a solution for computing one AXp/CXp.

Recent years witnessed a rapid development of logic-based explainability, with practically efficient solutions devised for a growing number of ML models. Overviews of these results are available [16, 18].

3 Explanations for Monotonic Classifiers

In [17] the authors have proposed algorithms for the computation of explanation of black-box monotonic classifier. The algorithm 1 (resp. 2) compute the abductive (resp. contrastive) explanations of the prediction of a feature $\mathbf{v}$, for a classifier κ with a set a features $\mathbb{F}$. For all features $i \in \mathbb{F}$ the value $\mathbf{v}_i$ must be bounded between $\lambda(i)$ and $\nu(i)$.

Algorithm 1 findAXp $\mathbb{F}$ $\mathbf{v}$

$\mathbf{v}_l \leftarrow (\mathbf{v}_1, ..., \mathbf{v}_N)$
$\mathbf{v}_u \leftarrow (\mathbf{v}_1, ..., \mathbf{v}_N)$
$(C, D, P) \leftarrow (\mathbb{F}, \emptyset, \emptyset)$
for all $i \in \mathbb{F}$ **do**
  $(\mathbf{v}_l, \mathbf{v}_u, C, D) \leftarrow FreeAttr(i, \mathbf{v}, \mathbf{v}_l, \mathbf{v}_u, C, D)$
  if $\kappa(\mathbf{v}_l) \neq \kappa(\mathbf{v}_u)$ **then**
    $(\mathbf{v}_l, \mathbf{v}_u, D, P) \leftarrow FixAttr(i, \mathbf{v}, \mathbf{v}_l, \mathbf{v}_u, C, D)$
  endif
endfor
return P

Algorithm 2 findCXp $\mathbb{F}$ **v**

$\mathbf{v}_l \leftarrow (\lambda(1), ..., \lambda(N))$
$\mathbf{v}_u \leftarrow (\nu(1), ..., \nu(N))$
$(C, D, P) \leftarrow (\mathbb{F}, \emptyset, \emptyset)$
for all $i \in \mathbb{F}$ **do**
  $(\mathbf{v}_l, \mathbf{v}_u, C, D) \leftarrow FixAttr(i, v, \mathbf{v}_l, \mathbf{v}_u, C, D)$
  if $\kappa(\mathbf{v}_l) = k(\mathbf{v}_u)$ **then**
    $(\mathbf{v}_l, \mathbf{v}_u, D, P) \leftarrow FreeAttr(i, \mathbf{v}, \mathbf{v}_l, \mathbf{v}_u, C, D)$
  endif
endfor
return P

Algorithm 3 FreeAttr i **v** $\mathbf{v}_l$ $\mathbf{v}_u$ A B

$\mathbf{v}_l \leftarrow (\mathbf{v}_{l1}, ...\lambda(i), ..., \mathbf{v}_{lN})$
$\mathbf{v}_l \leftarrow (\mathbf{v}_{u1}, ...\nu(i), ..., \mathbf{v}_{uN})$
$(A, B) \leftarrow (A \setminus \{i\}, B \cup \{i\})$
return (v_l, v_u, A, B)

Algorithm 4 FixAttr i **v** $\mathbf{v}_l$ $\mathbf{v}_u$ A B

$\mathbf{v}_l \leftarrow (\mathbf{v}_{l1}, ...\mathbf{v}_i, ..., \mathbf{v}_{lN})$
$\mathbf{v}_l \leftarrow (\mathbf{v}_{u1}, ...\mathbf{v}_i, ..., \mathbf{v}_{uN})$
$(A, B) \leftarrow (A \setminus \{i\}, B \cup \{i\})$
return $(\mathbf{v}_l, \mathbf{v}_u, A, B)$

The idea is to go through the features one by one to see if they had an impact on the decision. For abductive explanation, two points in feature space are tested with min and max values on the tested feature. If the classifications are different, this feature has an impact on the answer and therefore answers the question "Why?" (so is kept in the explanation). For contrastive explanation, two points in feature space are tested with min and max values on all features except the tested feature. If the classification is the same and since the classifier is not trivial, it has to be changed to change the classification value and thus answers the question "Why not?" (so is kept in the explanation). For pedagogical reasons, this explanation is simplified : the two points in feature space tested take into account the responses for the features previously tested.

4 Proofs

The aim of the work is threefold:

- prove that the algorithms are correct
- extract confidence code from the proof of correctness
- check if the monotonicity constraint of the classifiers can be relaxed

To achieve these objectives we use a proof assistant that allows code extraction: Coq [5].

4.1 Coq

Coq is a proof assistant that is built on a programming language called Gallina that allows for the expression of mathematical theorems and software specifications. This language combines higher-order logic and functional programming. Coq provides a command language for defining functions and predicates, stating theorems and specifications, and developing formal proofs interactively. The proofs can then be checked by a small certification "kernel". Additionally, Coq allows for the extraction of certified programs into languages such as OCaml, Haskell, or Scheme.

4.2 The Coq formalization

To be Coq compatible, it is necessary to alter the method used to encode the original algorithms (algorithms 1, 2 3 and 4) while ensuring that the writing facilitated the proof of termination and correction and the code generation.

As in Coq the data structures are homogeneous, all the features should have the same type T ie for all i in $\mathbb{F}$: $\mathcal{D}_i = T$. This type T need a total order relation. This may seem restrictive, but all digital types, for example, fulfill these constraints. A point in feature space is represented by a list of elements of type T. The return of the classifier is of type Tk. The only requirement on Tk

[5] `https://coq.inria.fr`

is the existence of a decidable equality (Tk_eq_dec). C and D are not used and are dropped, while P is coded by a list of naturals (indexes of the features).

The original iterative way of coding the algorithms is replaced by a recursive version more in line with the Coq environment. For abductive (resp. contrastive) explanation, two auxiliary functions are required: findAXp_aux which add parameters that corresponds to the variables of the original algorithm and findAXp_aux_j that corresponds to the loop. To ease the proof termination for Coq, the parameter j that indicates the feature to analyze, is chosen to be strictly decreasing.

The Coq version of the algorithm 1 is given in algorithm 5. An equivalent transformation is done for findCXp.

Algorithm 5 findAXp κ **v**

```
(* Find the abductive explanation of v *)
(* j : (N-j) the feature to check *)
(* p : the feature before (N-j) that are part of the explanation *)
(* j is decreasing for Coq to proof the termination *)
Fixpoint findAXp_aux_j (k: list T → Tk) (j:nat) (v vl vu: list T) (p:list nat)
{struct j}: list nat :=
match j with
| 0 ⇒ p
| S jminus1 ⇒
  let '(nvl,nvu) := freeAttr (N−j) vl vu in
  match T_eq_dec (k nvl) (k nvu) with
  | false ⇒ let '(nvl,nvu,np) := fixAttr (N−j) v nvl nvu p in
              findAXp_aux_j k jminus1 v nvl nvu np
  | true ⇒ findAXp_aux_j k jminus1 v nvl nvu p
    end
end.

(* Find the abductive explanation of v *)
(* i : the feature to check *)
(* p : the feature before i that are part of the explanation *)
Definition findAXp_aux (k: list T → Tk) (i:nat) (v vl vu: list T) (p:list nat):
list nat :=
  findAXp_aux_j k (N−i) v vl vu p.

(* Find the abductive explanation of v *)
Program Definition findAXp (k: list T → Tk) (v: list T) : list nat :=
  findAXp_aux k 0 v v v nil.
```

For FreeAttr an auxiliary function is also needed. Coq version of the algorithm 3 is given in algorithm 6. Equivalent transformation are done for FixAttr.

The Coq formalization of the AXp property (Equation 1) is given in algorithm 7. Equivalent formalization are done for CXp property (Equation 2).

Algorithm 6 FreeAttr i $\mathbf{v}_l$ $\mathbf{v}_u$

```
(* Replace the i-th elements of the list vl and vu
 by a value determined by n *)
Fixpoint freeAttr_aux (i:nat) (n:nat) (vl:list T) (vu:list T) :=
   match i,vl,vu with
   | 0,_::ql,_::qu ⇒ ((lambda n)::ql,(nu n)::qu)
   | _,tl::ql,tu::qu ⇒ let (rl,ru) := freeAttr_aux (i−1) n ql qu
                       in (tl::rl,tu::ru)
   |_,_,_ ⇒ (vl,vu)
   end.

(* Replace the i-th elements of the lists vl and vu
 by lambda i and nu i *)
Definition freeAttr (i:nat) (vl:list T) (vu:list T) := freeAttr_aux i i vl vu.
```

Algorithm 7 is_AXp

```
Definition is_weak_AXp (k : list T → Tk) (v: list T) (p:list nat) : Prop :=
   forall (x: list T),
   List.length v = N
   (* x in feature space *)
∧ List.length x = N
   (* the values of the features of x are in the bounds *)
∧ (forall (j:nat), j>=0 ∧ j< N
      → (led (lambda j) (get j x) ∧ led (get j x) (nu j)))
   (* the values of the feature constraints in the explanation
      are the same in x and v *)
∧ (forall (j:nat), j>=0 ∧ j< N
      → ((mem j p ∧ get j x = get j v) ∨ (not (mem j p))))
→ k(x)=k(v).

Definition is_AXp (k : list T → Tk) (v: list T) (p:list nat) : Prop :=
   (* satisfy the equation of AXp *)
   is_weak_AXp k v p
∧ (* no subset satisfies the equation of AXp *)
   forall (q:list nat), (is_strict_subset q p) → not (is_weak_AXp k v q).
```

4.3 Results

In [17] the algorithms were supposed correct for monotonic classifiers. The constraints on the classifier is relaxed for a more general class of classifiers : the *stable* classifiers.

The AXp finder algorithm is proved correct for stable classifiers.

Theorem 1.

$$\begin{aligned}\forall k : (list\ T \to Tk), \mathsf{stable}\ k \to (&\\ &\forall v : list\ T, length\ v = N\\ &\wedge \forall j \in [0, N[, \lambda(j) \leq v_j \leq \nu(j)\\ &\to \mathsf{is\_AXp}\ k\ v\ (\mathsf{findAXp}\ k\ v)\\ &)\end{aligned}$$

Proof. Done with Coq. □

The CXp finder algorithm is proved correct for stable and nontrivial classifiers.

Theorem 2.

$$\begin{aligned}\forall k : (list\ T \to Tk), \mathsf{not\_trivial}\ k \wedge \mathsf{stable}\ k \to (&\\ &\forall v : list\ T, length\ v = N\\ &\wedge \forall j \in [0, N[, \lambda(j) \leq v_j \leq \nu(j)\\ &\to \mathsf{is\_CXp}\ v\ (\mathsf{findCXp}\ v)\\ &)\end{aligned}$$

Proof. Done with Coq. □

4.4 Proof sketch

For each algorithm, ten invariants are identified (figure 1). Each of the invariants is proven true at the start (lemmas pre_post_findAXp et pre_post_findCXp). Each of the invariants is proven preserved by the two recursive cases of the two algorithms (forty lemmas preserveRXCasY_AXp and preserveRXCasY_CXp). Lemmas pre_post_findAXp_aux and pre_post_findCXp_aux proves that the ten invariants implies three post-conditions (figure 2). Eventually, axp_all and cxp_all proves that the three post-conditions in the terminal case implies the AXp and CXp properties.

The forty lemma of preservation of the invariant need lemmas on FreeAttr and FixAttr like preservation of the list size, preservation of the bounded properties of the features, only the i-th element is modify,... At the end 137 lemmas are needed to prove the two theorems of correctness of the algorithms.

	findAXp_aux $\kappa\ i\ \mathbf{v}\ \mathbf{v}_l\ \mathbf{v}_u\ p$	findCXp_aux $\kappa\ i\ \mathbf{v}\ \mathbf{v}_l\ \mathbf{v}_u\ p$
R_1	$\forall j \in [0, N[, \lambda(j) \leq \mathbf{v}_j \leq \nu(j)$ $\forall j \in [0, N[, \lambda(j) \leq \mathbf{v}_{lj} \leq \nu(j)$ $\forall j \in [0, N[, \lambda(j) \leq \mathbf{v}_{uj} \leq \nu(j)$	idem
R_1	$\kappa\ \mathbf{v}_l = \kappa\ \mathbf{v}_u = \kappa\ \mathbf{v}$	$\kappa\ \mathbf{v}_l \neq \kappa\ \mathbf{v}_u$
R_3	$i \geq 0$	idem
R_4	$\forall j \in [0, N[, \lambda(j) = \mathbf{v}_{lj} \wedge \nu(j) = \mathbf{v}_{uj}$ $\vee \mathbf{v}_j = \mathbf{v}_{lj} \wedge \mathbf{v}_j = \mathbf{v}_{uj}$	not needed
R_5	$\forall j \in [0, N[, j \geq i \rightarrow j \notin p$	idem
R_6	$\forall j \in [0, N[, j \in p \rightarrow \mathbf{v}_j = \mathbf{v}_{lj} \wedge \mathbf{v}_j = \mathbf{v}_{uj}$	$\forall j \in [0, N[, j \in p \rightarrow \lambda(j) = \mathbf{v}_{lj} \wedge \nu(j) = \mathbf{v}_{uj}$
R_7	$\forall j \in [0, N[, j < i \wedge j \notin p \rightarrow \lambda(j) = \mathbf{v}_{lj} \wedge \nu(j) = \mathbf{v}_{uj}$	$\forall j \in [0, N[, j \geq i \rightarrow \lambda(j) = \mathbf{v}_{lj} \wedge \nu(j) = \mathbf{v}_{uj}$
R_8	is_sorted p	idem
R_9	$\forall j \in [0, N[, j \geq i \rightarrow \mathbf{v}_j = \mathbf{v}_{lj} \wedge \mathbf{v}_j = \mathbf{v}_{uj}$	$\forall j \in [0, N[, j < i \wedge j \notin p \rightarrow \mathbf{v}_j = \mathbf{v}_{lj} \wedge \mathbf{v}_j = \mathbf{v}_{uj}$
R_{10}	$\forall x, x_0, x_1, p = x_0@(x :: x_1) \rightarrow \exists nv_l, nv_u,$ $(\forall j \in [0, N[,$ $((p \in x_1 \vee j > x) \wedge \mathbf{v}_j = nv_{lj} \wedge \mathbf{v}_j = nv_{uj})$ $\vee(\neg(p \in x_1 \vee j > x) \wedge \lambda(j) = nv_{lj} \wedge \nu(j) = nv_{uj}))$ $\wedge \kappa\ nv_l \neq \kappa\ nv_u$	$\forall x, x_0, x_1, p = x_0@(x :: x_1) \rightarrow \exists nv_l, nv_u,$ $(\forall j \in [0, N[,$ $((p \in x_1 \vee j > x) \wedge \lambda(j) = nv_{lj} \wedge \nu(j) = nv_{uj})$ $\vee(\neg(p \in x_1 \vee j > x) \wedge \mathbf{v}_j = nv_{lj} \wedge \mathbf{v}_j = nv_{uj}))$ $\wedge \kappa\ nv_l = \kappa\ nv_u$

Fig. 1: Invariants

	findAXp_aux $\kappa\ i\ \mathbf{v}\ \mathbf{v}_l\ \mathbf{v}_u\ p$	findCXp_aux $\kappa\ i\ \mathbf{v}\ \mathbf{v}_l\ \mathbf{v}_u\ p$
E_1	is_weak_AXp $\kappa\ \mathbf{v}$ (findAXp_aux $\kappa\ i\ \mathbf{v}\ \mathbf{v}_l\ \mathbf{v}_u\ p$)	is_weak_CXp $\kappa\ \mathbf{v}$ (findCXp_aux $\kappa\ i\ \mathbf{v}\ \mathbf{v}_l\ \mathbf{v}_u\ p$)
E_2	is_sorted (findAXp_aux $\kappa\ i\ \mathbf{v}\ \mathbf{v}_l\ \mathbf{v}_u\ p$)	idem
E_3	$\forall x, x_0, x_1, p,$ (findAXp_aux $\kappa\ i\ \mathbf{v}\ \mathbf{v}_l\ \mathbf{v}_u\ p) = x_0@(x :: x_1)$ $\rightarrow \exists nv_l, nv_u : (\forall j \in [0, N[,$ $((p \in x_1 \vee j > x) \wedge \mathbf{v}_j = nv_{lj} \wedge \mathbf{v}_j = nv_{uj})$ $\vee(\neg(p \in x_1 \vee j > x) \wedge \lambda(j) = nv_{lj} \wedge \nu(j) = nv_{uj}))$ $\wedge \kappa\ nv_l \neq \kappa\ nv_u$	$\forall x, x_0, x_1, p,$ (findCXp_aux $\kappa\ i\ \mathbf{v}\ \mathbf{v}_l\ \mathbf{v}_u\ p) = x_0@(x :: x_1)$ $\rightarrow \exists nv_l, nv_u : (\forall j \in [0, N[,$ $((p \in x_1 \vee j > x) \wedge \lambda(j) = nv_{lj} \wedge \nu(j) = nv_{uj})$ $\vee(\neg(p \in x_1 \vee j > x) \wedge \mathbf{v}_j = nv_{lj} \wedge \mathbf{v}_j = nv_{uj}))$ $\wedge \kappa\ nv_l = \kappa\ nv_u$

Fig. 2: Post conditions

4.5 Computing explanations that are guaranteed to be correct

Let's recall that the objective of the work is to compute explanations that are guaranteed to be correct. Coq is used to extract OCaml code that is certified correct, i.e. it validates proven properties. We can provide a certified tool that computes correct explanations.

The generated OCaml code is not modified, it is just reorganized. The *.mli* file generated by Coq is converted into a signature module (with some additional functions). The *.ml* file generated by Coq is converted into a functor (conformed with the signature module) parameterized by a dataset. For each dataset the axioms (N, T, Tk, `Tk_eq_dec`, λ and ν) need to be realized. k is implemented using the pyml librairy [6] which allow to make an efficient binding between the OCaml generated by Coq and the Python used to generated and run classifiers.

5 Experiments

In this section, we will present a series of experiments aimed at demonstrating the feasibility and scalability of our approach. Through these experiments, we will illustrate how our method can address the challenges we have identified and provide tangible evidence of its potential benefits.

Monotonic vs stable The literature commonly employs the monotonicity constraint more frequently than the stability constraint. However, even for monotonicity, it is challenging to locate tools that generate models that ensure this property. As far as we are aware, no tool exists that guarantees model stability. As a result, our experiments are conducted on models that are (supposedly) monotonic.

5.1 Dataset

Four dataset are used from `https://www.kaggle.com` :

- Car Evaluation Data Set [7]
- Heart Failure Prediction [8]
- Placement data full class [9]
- The Complete Pokemon Dataset [10]

Some modifications (feature digitization, remove non digital features, remove incomplete instances, modification of some features of output for some instances,. . .) are needed to make the datasets monotonic.

A summary of the dataset properties is given in figure 3.

[6] `https://github.com/thierry-martinez/pyml`
[7] `https://www.kaggle.com/datasets/elikplim/car-evaluation-data-set`
[8] `https://www.kaggle.com/datasets/andrewmvd/heart-failure-clinical-data`
[9] `https://www.kaggle.com/datasets/barkhaverma/placement-data-full-class`
[10] `https://www.kaggle.com/datasets/rounakbanik/pokemon`

Name	nb instances	nb features	nb classes
Heart	299	11	2
Car	1728	6	4
Placement	148	7	4
Pokemon	800	31	2

Fig. 3: Properties of the datasets

5.2 Models

XGBoost[11] [3] is utilized to create models that exhibit monotonicity. After generating the model, a Python program is used to perform a test to ensure that its predictions on the dataset adhere to both the monotonicity and stability constraints. Although XGBoost claims to ensure monotonicity, it may not hold true for certain datasets. However, since the predictions are stable, the models can still be used to benchmark the code generated by Coq.

Name	accuracy	monotonic ?	stable ?	size of the json file
Heart	71.72%	No	Yes	59.9Kio
Car	99.12%	Yes	Yes	650.9Kio
Placement	63.27%	Yes	Yes	19.0 Kio
Pokemon	98.86%	Yes	Yes	48.1 Kio

Fig. 4: Properties of the models generated by XGBoost

A summary of the model's properties is given in figure 4.

5.3 Computing explanations

For using the certified code (see section 4.5), for each of the dataset, the axioms N, T, Tk, `Tk_eq_dec`, λ and ν need to be realized. N is givent in figure 3. For the four dataset T and Tk are $float$ and `Tk_eq_dec` the equality on $float$. λ and ν are extracted from the dataset.

The experiment are run in a Dell Inc. Latitude 7400 with 32Gio RAM and 8 Intel® Core™ i7-8665U CPU @ 1.90GHz. The operating system is Ubuntu 22.04.2 LTS (64 bits). A summary of the explanations properties is given in figure 5.

As can be observed, the explanation sizes are in general small, which confirms the interest of computing AXp and CXp explanation and helps to explain the model.

[11] `https://xgboost.readthedocs.io/en/stable/`

Name	overage size AXp	overage size CXp	time to compute AXp and CXp in the entire dataset
Heart	2.45	1.61	real 0m3,665s user 0m24,991s sys 0m0,909s
Car	2.39	1.27	real 0m54,383s user 6m1,489s sys 0m1,704s
Placement	2.14	1.51	real 0m1,603s user 0m8,493s sys 0m0,864s
Pokemon	1.71	2.76	real 0m21,740s user 2m47,066s sys 0m1,600s

Fig. 5: Properties of the explanations

The algorithm makes $2*N$ calls to κ, which is linear in the number of features. This enables the code to scale effectively with larger feature sets. The number of feature explain the difference of computation times for the different models. Even if the *car* models has only six features, the execution time is higher than expected. We assume that it is due to the size of the generated model. Despite this, the (real) computation time for calculating explanations of several hundred instances does not exceed one minute, which is reasonable considering that the code is written for certification rather than optimization.

5.4 Comparison with Shap

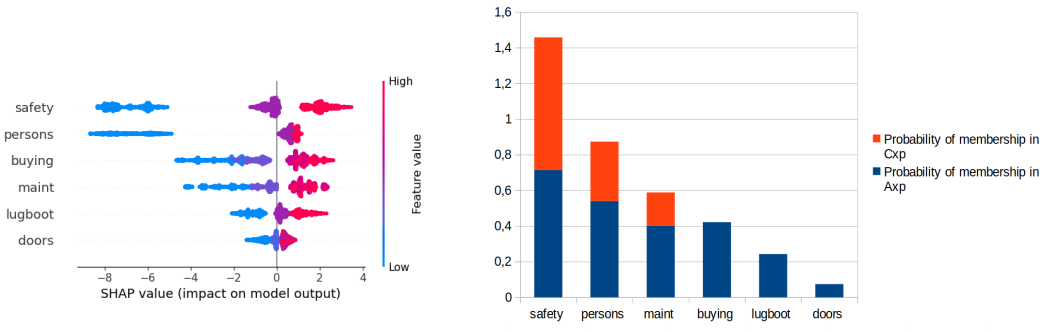
Shap [15] is a model-agnostic approaches who's goal is to explain the output of any machine learning model. There are recent negative results on the tractability of exact SHAP learning [2] and on the adequacy of Shapley values for explainability [9].

As our explanations are guaranteed to be correct, any discrepancies between our results and those of Shap will favor theses negative results.

For the car dataset, Shap highlights the features that are the more used in the explanations generated by the certified tool (see figure 6).

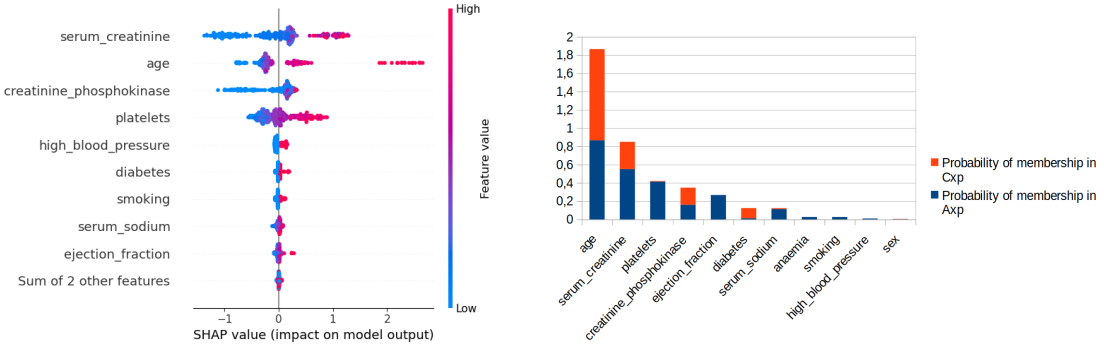
For the heart dataset, Shap does not highlith the *ejection fraction* feature, that is used in which is present in almost 30% of AXp (see figure 7).

Conversely, for the placement and pokemon dataset, Shap highlith some feature that are hardly present in the explanations generated by the certified tool. For example the *mba_p* feature for the placement dataset (see figure 8).



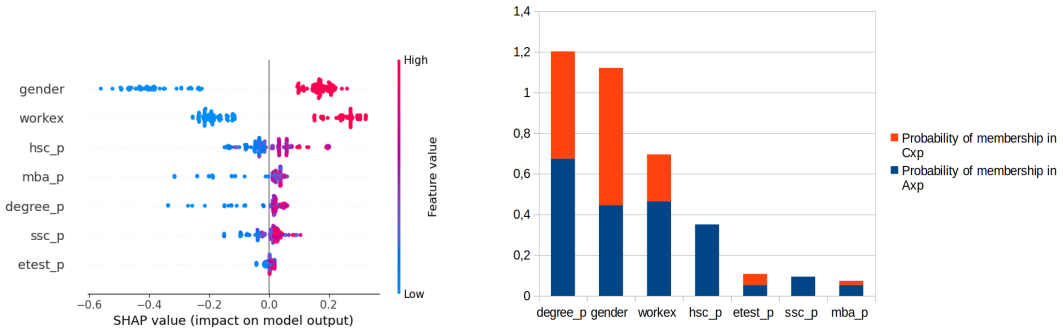
(a) Shap (b) Certified algorithm

Fig. 6: *Car* model and dataset



(a) Shap (b) Certified algorithm

Fig. 7: *Heart* model and dataset



(a) Shap (b) Certified algorithm

Fig. 8: *Placement* model and dataset

6 Conclusions

TO DO

References

1. Audemard, G., Koriche, F., Marquis, P.: On tractable XAI queries based on compiled representations. In: KR. pp. 838–849 (2020)
2. den Broeck, G.V., Lykov, A., Schleich, M., Suciu, D.: On the tractability of SHAP explanations. J. Artif. Intell. Res. **74**, 851–886 (2022). https://doi.org/10.1613/jair.1.13283, https://doi.org/10.1613/jair.1.13283
3. Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 785–794. KDD '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2939672.2939785, http://doi.acm.org/10.1145/2939672.2939785
4. Daniels, H., Velikova, M.: Monotone and partially monotone neural networks. IEEE Trans. Neural Networks **21**(6), 906–917 (2010)
5. Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A survey of methods for explaining black box models. ACM Comput. Surv. **51**(5), 93:1–93:42 (2019)
6. Gunning, D.: Explainable artificial intelligence (xai). https://www.darpa.mil/attachments/DARPA-BAA-16-53.pdf (2016), dARPA-BAA-16-53
7. Gunning, D., Aha, D.W.: Darpa's explainable artificial intelligence (XAI) program. AI Mag. **40**(2), 44–58 (2019). https://doi.org/10.1609/aimag.v40i2.2850, https://doi.org/10.1609/aimag.v40i2.2850
8. Gunning, D., Stefik, M., Choi, J., Miller, T., Stumpf, S., Yang, G.: XAI - explainable artificial intelligence. Sci. Robotics **4**(37) (2019). https://doi.org/10.1126/scirobotics.aay7120, https://doi.org/10.1126/scirobotics.aay7120
9. Huang, X., Marques-Silva, J.: The inadequacy of shapley values for explainability. CoRR **abs/2302.08160** (2023). https://doi.org/10.48550/arXiv.2302.08160, https://doi.org/10.48550/arXiv.2302.08160
10. Ignatiev, A.: Towards trustable explainable AI. In: IJCAI. pp. 5154–5158 (2020)
11. Ignatiev, A., Narodytska, N., Asher, N., Marques-Silva, J.: From contrastive to abductive explanations and back again. In: AIxIA. pp. 335–355 (2020)
12. Ignatiev, A., Narodytska, N., Marques-Silva, J.: Abduction-based explanations for machine learning models. In: AAAI. pp. 1511–1519 (2019)
13. Ignatiev, A., Narodytska, N., Marques-Silva, J.: On validating, repairing and refining heuristic ML explanations. CoRR **abs/1907.02509** (2019), http://arxiv.org/abs/1907.02509
14. Lundberg, S.M., Lee, S.: A unified approach to interpreting model predictions. In: NeurIPS. pp. 4765–4774 (2017)
15. Lundberg, S.M., Lee, S.: A unified approach to interpreting model predictions. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA. pp. 4765–4774 (2017), https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html
16. Marques-Silva, J.: Logic-based explainability in machine learning. CoRR **abs/2211.00541** (2022). https://doi.org/10.48550/arXiv.2211.00541, https://doi.org/10.48550/arXiv.2211.00541
17. Marques-Silva, J., Gerspacher, T., Cooper, M.C., Ignatiev, A., Narodytska, N.: Explanations for monotonic classifiers. In: ICML. pp. 7469–7479 (2021)

18. Marques-Silva, J., Ignatiev, A.: Delivering trustworthy AI through formal XAI. In: AAAI. pp. 12342–12350 (2022)
19. Marques-Silva, J., Janota, M., Mencía, C.: Minimal sets on propositional formulae. problems and reductions. Artif. Intell. **252**, 22–50 (2017), https://doi.org/10.1016/j.artint.2017.07.005
20. Miller, T.: Explanation in artificial intelligence: Insights from the social sciences. Artif. Intell. **267**, 1–38 (2019)
21. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. **32**(1), 57–95 (1987)
22. Ribeiro, M.T., Singh, S., Guestrin, C.: "why should I trust you?": Explaining the predictions of any classifier. In: KDD. pp. 1135–1144 (2016)
23. Ribeiro, M.T., Singh, S., Guestrin, C.: Anchors: High-precision model-agnostic explanations. In: AAAI. pp. 1527–1535 (2018)
24. Seshia, S.A., Sadigh, D., Sastry, S.S.: Toward verified artificial intelligence. Commun. ACM **65**(7), 46–55 (2022). https://doi.org/10.1145/3503914, https://doi.org/10.1145/3503914
25. Shih, A., Choi, A., Darwiche, A.: A symbolic approach to explaining bayesian network classifiers. In: IJCAI. pp. 5103–5111 (2018)