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) [7–9] has witnessed massive interest [6]. Explainability has also been proposed as a core component of efforts for the verification of ML models [25].

Unfortunately, most work on XAI offers no guarantees of rigor. Model-agnostic XAI approaches [16, 23, 24] represent one such example. As a result of the lack of guarantees of rigor, there is by now comprehensive evidence [10, 11, 14] 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) [17, 19]. 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 [18]. 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 follow the notation and definitions used in earlier work [18].

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. $\kappa : \mathbb{F} \to \mathcal{K}$. Since we assume that κ is non-constant, then the ML classifier $\mathbb{M}$ is declared *nontrivial*, i.e. $\exists \mathbf{a}, \mathbf{b} \in \mathbb{F}, \kappa(\mathbf{a}) \neq \kappa(\mathbf{b})$.

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

[3] The paper adopts the classification of monotonic classifiers proposed in earlier work [5].

classifier, this signifies a fully monotonic classifier. In addition, the interaction with a classifier is restricted to computing 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.

As a monotonic classifier, we used a heart failure prediction in the section 5, which depends on age and certain medical measures such as diabetes and platelet count. It is natural to expect the classifier to exhibit monotonicity, i.e., as age or diabetes levels increase, the risk of heart failure should also increase.

***Stable* classifiers.** An ML classifier is *stable* if the associated classification function respects $\forall \mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{F}, \mathbf{a} \leq \mathbf{b} \leq \mathbf{c} \land \kappa(\mathbf{a}) = \kappa(\mathbf{c}) \Rightarrow \kappa(\mathbf{a}) = \kappa(\mathbf{b}) = \kappa(\mathbf{c})$. 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*. Not all stable classifiers are monotonic.

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.

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

Prime implicant (PI) explanations [26] 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 [21], which answers a 'Why Not?' question, i.e. which features can be changed to change the prediction. A formal definition of contrastive explanation is proposed in recent work [12]. 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 [22], [12] proves a minimal hitting set (MHS) duality

[4] PI-explanations can be formulated as a problem of logic-based abduction, and so are also referred to as abductive explanations (AXp) [13]. More recently, AXp's have been studied from a knowledge compilation perspective [1].

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 [20]. 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 [17, 19].

3 Explanations for Monotonic Classifiers

In [18], the authors proposed algorithms for computing explanations of a black-box monotonic classifier. Algorithms 1 and 2 compute the abductive and contrastive explanations of the prediction of a feature $\mathbf{v}$, for a classifier κ with a set of features $\mathbb{F}$. For all features $i \in \mathbb{F}$, $\mathbf{v}_i$ must be bounded between $\lambda(i)$ and $\mu(i)$.

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**
  $(\mathbf{v}_l, \mathbf{v}_u, C, D) \leftarrow \mathsf{FreeAttr}(i, \mathbf{v}, \mathbf{v}_l, \mathbf{v}_u, C, D)$
  if $\kappa(\mathbf{v}_l) \neq \kappa(\mathbf{v}_u)$ **then**
    $(\mathbf{v}_l, \mathbf{v}_u, D, P) \leftarrow \mathsf{FixAttr}(i, \mathbf{v}, \mathbf{v}_l, \mathbf{v}_u, C, P)$
  endif
endfor
return P

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

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

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

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

The idea behind the algorithms is to analyze the features one by one and determine whether they have an impact on the decision. In abductive explanations, two points in the feature space are tested with the minimum and maximum

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

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

values for the feature being tested. If the classifications are different, then that feature has an impact on the answer and is included in the explanation to answer the question "Why?". In contrastive explanations, two points in the feature space are tested with the minimum and maximum values for all features except the one being tested. If the classification is the same, and since the classifier is not trivial, the feature must be changed to alter the classification value, answering the question "Why not?" and is included in the explanation. For pedagogical reasons, this explanation is simplified: in the algorithm, the features are tested while considering the responses obtained for the features previously tested.

4 Proofs

The aim of this work is threefold: to prove the correctness of the algorithms, to extract the confidence code from the proof of correctness, and to investigate whether the monotonicity constraint of the classifiers can be relaxed.

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

The Coq proof, the Python codes used to generate and check the models, and the OCaml code used to run the experiments (section 5) are available at this link: https://github.com/hurault/tap23.

4.1 Coq

Coq is a proof assistant that is built on a programming language called Gallina, which allows expressing 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 the extraction of certified programs into languages such as OCaml, Haskell, or Scheme.

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

4.2 The Coq formalization

To ensure compatibility with Coq, it was necessary to modify the method used to encode the original algorithms (1, 2, 3, and 4), while ensuring that the changes facilitated the proof of termination and correctness, as well as code generation.

In Coq, the data structures are homogeneous, so all features should have the same type T. That is, for all i in $\mathbb{F}$, $\mathcal{D}_i = T$. This type T requires a total order relation. This may seem restrictive, but all digital types, for example, meet these constraints. A point in feature space is represented by a list of elements of type T. The output of the classifier has type Tk. The only requirement on Tk 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 has been replaced by a recursive version that is more in line with the Coq environment. For the abductive (resp. contrastive) explanation, two auxiliary functions are required: findAXp_aux which adds parameters that correspond to the variables of the original algorithm and findAXp_aux_j that corresponds to the loop. To ease the proof of termination for Coq, the parameter j that indicates the feature to analyze is chosen to be strictly decreasing.

The Coq version of algorithm 1 is presented in algorithm 5. A similar transformation has been carried out for findCXp.

An auxiliary function is also needed for FreeAttr. The Coq version of the original algorithm 3 is given in algorithm 6. Equivalent transformations are also done for FixAttr.

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

The formalization of the algorithms and their Coq proof can be found in the file Coq/AXp_CXp_Stable_nfeatures.v.

4.3 Results

In [18], the authors proposed algorithms and argued for their correctness for monotonic classifiers. In our paper, we provide a proof of their correctness and relax the monotonicity constraint on the classifier to a more general class of classifiers known as *stable* classifiers.

The AXp finder algorithm is proven correct for stable classifiers.

Theorem 1.

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

Proof. Done with Coq. □

Algorithm 5 findAXp κ $\mathbf{v}$

```
(* 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.
```

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,(mu 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 mu 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 *)
   (* led is the relation order in feature space *)
∧ (forall (j:nat), j>=0 ∧ j< N
      → (led (lambda j) (get j x) ∧ led (get j x) (mu 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).
```

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

Theorem 2.

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

Proof. Done with Coq. □

4.4 Proof sketch

Here is the proof sketch for the algorithm computing abductive explanations. The same structure is used for contrastive explanations.

1. A property R, depending on κ,i,$\mathbf{v}$,$\mathbf{v}_l$,$\mathbf{v}_u$ and p is identified and proven to be true for the initial values of findAXp_aux ie $R(\kappa, 0, \mathbf{v}, \mathbf{v}, \mathbf{v}, nil)$ (lemma R_init_Axp).
2. The property R is proven to be preserved by the two recursive cases of the algorithm:

(a) $R(\kappa, i, \mathbf{v}, \mathbf{v}_l, \mathbf{v}_u, p)$
$\wedge(\mathbf{nv}_l, \mathbf{nv}_u) = \mathsf{freeAttr}(i, \mathbf{v}, \mathbf{v}_l, \mathbf{v}_u, p)$
$\wedge(\mathbf{nnv}_l, \mathbf{nnv}_u, np) = \mathsf{fixAttr}(i, \mathbf{v}, \mathbf{nv}_l, \mathbf{nv}_u, p)$
$\rightarrow R(\kappa, i+1, \mathbf{v}, \mathbf{nnv}_l, \mathbf{nnv}_u, np)$

(b) $R(\kappa, i, \mathbf{v}, \mathbf{v}_l, \mathbf{v}_u, p)$
$\wedge(\mathbf{nv}_l, \mathbf{nv}_u) = \mathsf{freeAttr}(i, \mathbf{v}, \mathbf{v}_l, \mathbf{v}_u, p)$
$\rightarrow R(\kappa, i+1, \mathbf{v}, \mathbf{nv}_l, \mathbf{nv}_u, p)$

R is in fact a conjunction of several sub-properties, each of these R_i properties is handled by the lemmas preserveRiCas2_AXp and preserveRiCas3_AXp for case (a) and case (b).

3. A second property E, depending on κ,i,$\mathbf{v}$,$\mathbf{v}_l$,$\mathbf{v}_u$ and p is identified and proven to be implied by R i.e. $R(\kappa, i, \mathbf{v}, \mathbf{v}_l, \mathbf{v}_u, p) \rightarrow E(\kappa, i, \mathbf{v}, \mathbf{v}_l, \mathbf{v}_u, p)$ (lemma R_implies_E_findAXp).
4. The last step proves that E in the terminal case, i.e. $E(\kappa, N, \mathbf{v}, \mathbf{v}_l, \mathbf{v}_u, p)$, implies that p is an abductive explanation of $\mathbf{v}$ for κ (theorem axp_all).

R : The property R is a conjunction of ten or eleven properties that are listed in Figure 1. These properties are about the size of the lists used for representing an instance of the feature space (R_0) and the bounded properties of the elements of these lists (R_1). R_4, R_6, R_7, and R_9 explicitly state the values of $\mathbf{v}_l$ and $\mathbf{v}_u$ depending on i (the current feature), λ, μ, $\mathbf{v}$, and p. R_5 and R_8 give some information about p: it is sorted and can only contain features that have been reviewed. $R10$ explains the consequences for a feature x of being part of the explanation p. R_2 links the value of $\kappa(\mathbf{v})$, $\kappa(\mathbf{v}_l)$, and $\kappa(\mathbf{v}_u)$.

E : The property E is a conjunction of three properties that are listed in figure 2. E_1 states that the explanation is a weak explanation. E_2 states that the explanation is sorted and E_3 explains the consequences for a feature x of being part of an explanation. E_2 and E_3 are necessary to prove that the explanation is subset-minimal.

Additional lemmas The proof of preservation of R requires lemmas on FreeAttr and FixAttr, such as the preservation of list size, bounded properties of features, and modification of only the i-th element. In total, 137 lemmas are necessary to prove the two correctness theorems for the algorithms.

4.5 Computing explanations that are guaranteed to be correct

Let us recall that the objective of this work is to compute explanations that are guaranteed to be correct. We have proof that if a classifier is stable, the algorithms generate correct explanations. To have confidence in a tool's generated explanations, one must have confidence in the algorithm's implementation and in the stability of the classifiers.

	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_0	$\mathsf{List.length}\ \mathbf{v} = N$ $\mathsf{List.length}\ \mathbf{v}_l = N$ $\mathsf{List.length}\ \mathbf{v}_u = N$	idem
R_1	$\forall j \in [0, N[, \lambda(j) \leq \mathbf{v}_j \leq \mu(j)$ $\forall j \in [0, N[, \lambda(j) \leq \mathbf{v}_{lj} \leq \mu(j)$ $\forall j \in [0, N[, \lambda(j) \leq \mathbf{v}_{uj} \leq \mu(j)$	idem
R_2	$\kappa(\mathbf{v}_l) = \kappa(\mathbf{v}_u) = \kappa(\mathbf{v})$	$\kappa(\mathbf{v}_l) \neq \kappa(\mathbf{v}_u)$
R_3	$i \geq 0$	idem
R_4	$\forall j \in [0, N[,$ $\lambda(j) = \mathbf{v}_{lj} \wedge \mu(j) = \mathbf{v}_{uj}$ $\vee\ \mathbf{v}_j = \mathbf{v}_{lj} \wedge \mathbf{v}_j = \mathbf{v}_{uj}$	not needed
R_5	$\forall j \in [0, N[, j \geq i \rightarrow j \notin p$	idem
R_6	$\forall j \in [0, N[, j \in p \rightarrow$ $\mathbf{v}_j = \mathbf{v}_{lj} \wedge \mathbf{v}_j = \mathbf{v}_{uj}$	$\forall j \in [0, N[, j \in p \rightarrow$ $\lambda(j) = \mathbf{v}_{lj} \wedge \mu(j) = \mathbf{v}_{uj}$
R_7	$\forall j \in [0, N[, j < i \wedge j \notin p \rightarrow$ $\lambda(j) = \mathbf{v}_{lj} \wedge \mu(j) = \mathbf{v}_{uj}$	$\forall j \in [0, N[, j \geq i \rightarrow$ $\lambda(j) = \mathbf{v}_{lj} \wedge \mu(j) = \mathbf{v}_{uj}$
R_8	$\mathsf{is\_sorted}\ p$	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}	$\forall x, x_0, x_1, p = x_0 @ (x :: x_1) \rightarrow$ $\exists nv_l, nv_u, (\forall j \in [0, N[,$ $((p \in x_1 \vee j > x) \wedge$ $\mathbf{v}_j = nv_{l_j} \wedge \mathbf{v}_j = nv_{u_j})$ $\vee$ $(\neg(p \in x_1 \vee j > x) \wedge$ $\lambda(j) = nv_{l_j} \wedge \mu(j) = nv_{u_j}))$ $\wedge$ $\kappa(nv_l) \neq \kappa(nv_u)$	$\forall x, x_0, x_1, p = x_0 @ (x :: x_1) \rightarrow$ $\exists nv_l, nv_u, (\forall j \in [0, N[,$ $((p \in x_1 \vee j > x) \wedge$ $\lambda(j) = nv_{l_j} \wedge \mu(j) = nv_{u_j})$ $\vee$ $(\neg(p \in x_1 \vee j > x) \wedge$ $\mathbf{v}_j = nv_{l_j} \wedge \mathbf{v}_j = nv_{u_j}))$ $\wedge$ $\kappa(nv_l) = \kappa(nv_u)$

Fig. 1: Invariants

	findAXp_aux κ i $\mathbf{v}$ $\mathbf{v}_l$ $\mathbf{v}_u$ p
E_1	is_weak_AXp κ $\mathbf{v}$ (findAXp_aux κ i $\mathbf{v}$ $\mathbf{v}_l$ $\mathbf{v}_u$ p)
E_2	is_sorted (findAXp_aux κ i $\mathbf{v}$ $\mathbf{v}_l$ $\mathbf{v}_u$ p)
E_3	$\forall x, x_0, x_1, p, (\text{findAXp\_aux } \kappa\ i\ \mathbf{v}\ \mathbf{v}_l\ \mathbf{v}_u\ p) = x_0@(x :: x_1) \rightarrow \exists nv_l, nv_u :$ $(\forall j \in [0, N[, ((p \in x_1 \vee j > x) \wedge \mathbf{v}_j = nv_{l_j} \wedge \mathbf{v}_j = nv_{u_j})$ $\vee(\neg(p \in x_1 \vee j > x) \wedge \lambda(j) = nv_{l_j} \wedge \mu(j) = nv_{u_j}))$ $\wedge\ \kappa(nv_l) \neq \kappa(nv_u)$

	findCXp_aux κ i $\mathbf{v}$ $\mathbf{v}_l$ $\mathbf{v}_u$ p
E_1	is_weak_CXp κ $\mathbf{v}$ (findCXp_aux κ i $\mathbf{v}$ $\mathbf{v}_l$ $\mathbf{v}_u$ p)
E_2	is_sorted (findCXp_aux κ i $\mathbf{v}$ $\mathbf{v}_l$ $\mathbf{v}_u$ p)
E_3	$\forall x, x_0, x_1, p, (\text{findCXp\_aux } \kappa\ i\ \mathbf{v}\ \mathbf{v}_l\ \mathbf{v}_u\ p) = x_0@(x :: x_1) \rightarrow \exists nv_l, nv_u :$ $(\forall j \in [0, N[, ((p \in x_1 \vee j > x) \wedge \lambda(j) = nv_{l_j} \wedge \mu(j) = nv_{u_j})$ $\vee(\neg(p \in x_1 \vee j > x) \wedge \mathbf{v}_j = nv_{l_j} \wedge \mathbf{v}_j = nv_{u_j}))$ $\wedge\ \kappa(nv_l) = \kappa(nv_u)$

Fig. 2: Post conditions

Confidence on the implementation Coq is used to extract OCaml code that is certified as correct, i.e. it validates proven properties. We can provide a certified tool that computes correct explanations.

The generated OCaml code is not modified, but it is reorganized. The *.mli* file generated by Coq is converted into a signature module (with some additional functions), while the *.ml* file generated by Coq is converted into a functor that conforms with the signature module and is parameterized by a dataset (file OCaml/explain.ml). For each dataset, the axioms (N, T, Tk, Tk_eq_dec, λ, and μ) need to be realized (file OCaml/dataset.ml). k is implemented using the pyml library[6], which allows for an efficient binding between the OCaml generated by Coq and the Python used to generate and run classifiers (file Ocaml/runAXpCXp.ml).

Confidence on the stablity of the classifiers The literature commonly employs the monotonicity constraint more frequently than the stability constraint. As far as we are aware, no tool exists that guarantees model stability while several tools offer ways to force monotony : XGBoost[7] [3], COMET[8] [27], Deep Lattice Network (DLN)[9] [28] or Certified Monotonic Neural Networks[10] [15].

None of those tools is certified as correct, so we have developed a Python program, using XGBoost, that check the stability (file XGBoost/verif_stable.py)

[6] https://github.com/thierry-martinez/pyml
[7] https://xgboost.readthedocs.io/en/stable/tutorials/monotonic.html
[8] https://github.com/AishwaryaSivaraman/COMET
[9] https://www.tensorflow.org/lattice/overview
[10] https://github.com/gnobitab/CertifiedMonotonicNetwork

and monotonicity (file XGBoost/verif_mono.py) of a classifier through **exhaustive testing** on the predictions of the training and testing datatsets. In the next section, it will be shown that three out of the four models used are monotonic, but one is not. However, since the predictions are stable, the model can still be used.

5 Experiments

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

5.1 Datasets

Four datasets are used from https://www.kaggle.com and can be found in the XGBoost/dataset repository:

- Car Evaluation Data Set [11]
- Heart Failure Prediction [12]
- Placement data full class [13]
- The Complete Pokemon Dataset [14]

Some modifications, such as feature digitization, removal of non-digital features, removal of incomplete instances, and modification of some output features for certain instances, are done to make the datasets monotonic.

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

A summary of the properties of the datasets is presented in Figure 3.

5.2 Models

XGBoost[15] [3] is used to create models that exhibit monotonicity. The Python code can be found in the file XGBoost/build_mono.py. As explain previously,

[11] https://www.kaggle.com/datasets/elikplim/car-evaluation-data-set
[12] https://www.kaggle.com/datasets/andrewmvd/heart-failure-clinical-data
[13] https://www.kaggle.com/datasets/barkhaverma/placement-data-full-class
[14] https://www.kaggle.com/datasets/rounakbanik/pokemon
[15] https://xgboost.readthedocs.io/en/stable/

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.

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 properties of the models is presented in Figure 4.

5.3 Computing explanations

To use the certified code (see section 4.5), the axioms N, T, Tk, `Tk_eq_dec`, λ, and μ need to be implemented for each dataset. The N axiom is provided in figure 3. For the four datasets $T = float$, $T_k = int$, and `Tk_eq_dec` is the equality on int. λ and μ are extracted from the datasets.

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). The execution time is compute using the Linux `time` command.

Name	Average size AXp	Average size CXp	Time for the entire dataset	Time for 100 instances (proportion)
Heart	2.45	1.61	user 0m24.991s sys 0m0.909s	0m8.358s 0m0.304s
Car	2.39	1.27	user 6m1.489s sys 0m1.704s	0m20.919s 0m0.098s
Placement	2.14	1.51	user 0m8.493s sys 0m0.864s	0m8.739s 0m0.584s
Pokemon	1.71	2.76	user 2m47.066s sys 0m1.600s	0m20.883 0m0.200s

Fig. 5: Properties of the explanations

A summary of the explanation properties is given in Figure 5. As can be observed, the explanation sizes are generally small, which confirms the interest in computing AXp and CXp explanations and helps to explain the model.

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 features explains the difference in computation times for the different models and in particular why the execution time, reduced to 100 instances, is more important for the *pokemon* dataset which has three times more features than the others. Even if the *car* model has only six features, the execution time is higher than expected. We assume that this is due to the size of the generated model. Despite this, the computation time for calculating explanations of several hundred instances is reasonable considering that the code is written for certification rather than optimization.

As an example, Figure 6 shows the distribution of features in the explanations for the heart dataset and model.

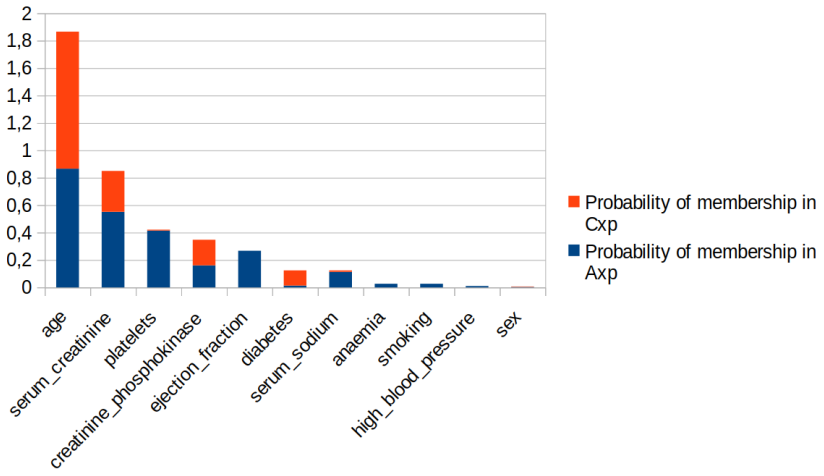

Fig. 6: Probability for a feature to be in the explanations

Our analysis reveals that, certain features are found to only contribute to the abductive explanations (*platelets*), while others are only relevant to the contrastive explanations (*diabetes*). This highlights the importance of distinguishing between these types of explanations when interpreting the classifier's behavior.

5.4 Comparison with direct mono-language implementation

To evaluate the effectiveness of the certified code, the algorithms were coded in Python (file XGBoost/find_xp.py) and run on the same models and dataset as the certified code. The comparison of the execution time is presented in the table in Figure 7.

Except for the model *car*, we can notice that the code generated by Coq takes around 25% more user time and 5% more system time. The difference in

Name		OCaml generated code	Python code	Additional time
Heart	user	0m24.991s	0m19.672s	27%
	sys	0m0.909s	0m0.880s	3%
Car	user	6m1.489s	1m2.110s	482%
	sys	0m1.704s	0m1.115s	52%
Placement	user	0m8.493s	0m6.712s	26.5%
	sys	0m0.864s	0m0.792s	9%
Pokemon	user	2m47.066s	2m15.932s	22.9%
	sys	0m1.600s	0m1.516s	5.5%

Fig. 7: Execution time comparison between the OCaml generated code and a Python code

execution time is due to the fact that the code was written to facilitate proof, not efficiency, and calls to classifiers are cross-language. However, despite this, the execution times are still reasonable and demonstrate the dual benefits of the work: proving that the algorithm is correct and providing a proof of concept for the use of certified correct code.

6 Conclusions

Explainability is posed to prove instrumental in delivering trustworthy AI, including in high-risk and safety-critical application domains. Unfortunately, informal XAI approaches offer no guarantees of rigor, and so their use in high-risk and safety-critical domains could be disastrous. There has been recent work on formal explainability, which offers guarantees of rigor in computed explanations. However, implemented algorithms (and some times their description) may not be correct.

This paper takes a first step towards delivering certified explanations. The paper considers monotonic classifiers, proves the correctness of proposed algorithms, and extracts confidence code from the proofs of correctness. The experimental results validate the scalability of the work.

Future work will extend the certification of formal explainability algorithms to classifiers more complex than the monotonic case. For tractable explainability problems, we envision adopting an approach similar to the one described in this paper. For more complex explainability problems, e.g. when the decision problems are (co-)NP-complete or even $\Sigma_2^{\mathrm{P}}/\Pi_2^{\mathrm{P}}$-hard, we envision adopting solutions similar to those used in the case of automated reasoners, especially SAT solvers [2, 4].

References

1. Audemard, G., Koriche, F., Marquis, P.: On tractable XAI queries based on compiled representations. In: KR. pp. 838–849 (2020)

2. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336. IOS Press (2021). https://doi.org/10.3233/FAIA336, https://doi.org/10.3233/FAIA336
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. Cruz-Filipe, L., Marques-Silva, J., Schneider-Kamp, P.: Formally verifying the solution to the boolean pythagorean triples problem. J. Autom. Reason. **63**(3), 695–722 (2019). https://doi.org/10.1007/s10817-018-9490-4, https://doi.org/10.1007/s10817-018-9490-4
5. Daniels, H., Velikova, M.: Monotone and partially monotone neural networks. IEEE Trans. Neural Networks **21**(6), 906–917 (2010)
6. 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)
7. Gunning, D.: Explainable artificial intelligence (xai). https://www.darpa.mil/attachments/DARPA-BAA-16-53.pdf (2016), dARPA-BAA-16-53
8. 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
9. 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
10. 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
11. Ignatiev, A.: Towards trustable explainable AI. In: IJCAI. pp. 5154–5158 (2020)
12. Ignatiev, A., Narodytska, N., Asher, N., Marques-Silva, J.: From contrastive to abductive explanations and back again. In: AIxIA. pp. 335–355 (2020)
13. Ignatiev, A., Narodytska, N., Marques-Silva, J.: Abduction-based explanations for machine learning models. In: AAAI. pp. 1511–1519 (2019)
14. 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
15. Liu, X., Han, X., Zhang, N., Liu, Q.: Certified monotonic neural networks. Advances in Neural Information Processing Systems **33** (2020)
16. Lundberg, S.M., Lee, S.: A unified approach to interpreting model predictions. In: NeurIPS. pp. 4765–4774 (2017)
17. 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
18. Marques-Silva, J., Gerspacher, T., Cooper, M.C., Ignatiev, A., Narodytska, N.: Explanations for monotonic classifiers. In: ICML. pp. 7469–7479 (2021)
19. Marques-Silva, J., Ignatiev, A.: Delivering trustworthy AI through formal XAI. In: AAAI. pp. 12342–12350 (2022)
20. 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

21. Miller, T.: Explanation in artificial intelligence: Insights from the social sciences. Artif. Intell. **267**, 1–38 (2019)
22. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. **32**(1), 57–95 (1987)
23. Ribeiro, M.T., Singh, S., Guestrin, C.: "why should I trust you?": Explaining the predictions of any classifier. In: KDD. pp. 1135–1144 (2016)
24. Ribeiro, M.T., Singh, S., Guestrin, C.: Anchors: High-precision model-agnostic explanations. In: AAAI. pp. 1527–1535 (2018)
25. 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
26. Shih, A., Choi, A., Darwiche, A.: A symbolic approach to explaining bayesian network classifiers. In: IJCAI. pp. 5103–5111 (2018)
27. Sivaraman, A., Farnadi, G., Millstein, T.D., den Broeck, G.V.: Counterexample-guided learning of monotonic neural networks. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual (2020), https://proceedings.neurips.cc/paper/2020/hash/8ab70731b1553f17c11a3bbc87e0b605-Abstract.html
28. You, S., Ding, D., Canini, K.R., Pfeifer, J., Gupta, M.R.: Deep lattice networks and partial monotonic functions. 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. 2981–2989 (2017), https://proceedings.neurips.cc/paper/2017/hash/464d828b85b0bed98e80ade0a5c43b0f-Abstract.html