Declarative Reasoning on Explanations Using Constraint Logic Programming

Laura State[1,2(✉)], Salvatore Ruggieri[1], and Franco Turini[1]

[1] University of Pisa, Pisa, Italy
laura.state@di.unipi.it
[2] Scuola Normale Superiore, Pisa, Italy

Abstract. Explaining opaque Machine Learning (ML) models is an increasingly relevant problem. Current explanation in AI (XAI) methods suffer several shortcomings, among others an insufficient incorporation of background knowledge, and a lack of abstraction and interactivity with the user. We propose REASONX, an explanation method based on Constraint Logic Programming (CLP). REASONX can provide declarative, interactive explanations for decision trees, which can be the ML models under analysis or global/local surrogate models of any black-box model. Users can express background or common sense knowledge using linear constraints and MILP optimization over features of factual and contrastive instances, and interact with the answer constraints at different levels of abstraction through constraint projection. We present here the architecture of REASONX, which consists of a Python layer, closer to the user, and a CLP layer. REASONX's core execution engine is a Prolog meta-program with declarative semantics in terms of logic theories.

1 Introduction

Artificial Intelligence (AI) systems are increasingly being adopted for taking critical decisions impacting society, such as loan concession in bank systems. The acceptance and trust of applications based on AI is hampered by the opaqueness and complexity of the Machine Learning (ML) models adopted, possibly resulting in biased or socially discriminatory decision-making [33].

For these reasons, there has recently been a flourishing of proposals for explaining the decision rationale of ML models [18,27,29,31], coined eXplanation in AI (XAI) methods. These approaches lack sufficient abstraction for reasoning over the decision rationale of the ML model. By reasoning, we mean the possibility for the user to define any number of conditions over factual and contrastive instances, which would codify both background knowledge and what-if analyses, and then looking at answers at the symbolic and intensional level.

To close this gap, we present REASONX (*reason to explain*), an explanation tool built in two layers. The first is in Python, closer to users, where decision tree (DT) models and user queries are parsed and translated. The DT can be the ML model itself, or a surrogate of other ML models at global/local level. The

S. Gaggl et al. (Eds.): JELIA 2023, LNAI 14281, pp. 132–141, 2023.
https://doi.org/10.1007/978-3-031-43619-2_10

second is in Constraint Logic Programming (CLP), where embedding of DTs and background knowledge are reasoned about, using a Prolog meta-program.

We display an exemplary dialogue between a fictional user and reasonx below. It is situated in the context of a credit application scenario, i.e. the user is a person whose credit application has been rejected by an automated decision-making system. Please note that while the information content is exactly what reasonx can provide, we enhanced the dialogue by translating the interaction into natural language, to mimic better a realistic interaction.

USER: Can I see the rule that led to the denial of my credit application?
REASONX: Your credit application was rejected, because your income is lower than 60,000 EUR/year, and you still have to pay back the lease of your car.
USER: Ok. Can you present me two options that will lead to a change of the decision outcome? Please take into consideration that I need a credit of at least 10,000 EUR. I would like to see options that require as little change as necessary.
REASONX: You have the following two options: You pay back the lease on the car, or you increase your age by 10 years (from 35 to 45 years).
USER: The second option presented is a bit strange. I am wondering whether this is salient in the model. Can I please see the options to obtain credit for an individual with the same properties as me, for a credit of at least 10,000 EUR, but with the feature age at 35 years or less (i.e. young applicant), instead of fixed?
REASONX: For the given profile, the credit is always rejected.
USER: Given this profile, how can the decision reversed, under as little changes as possible?
REASONX: Credit can be obtained, if the feature age is set to higher than 35 years.
USER: This interesting and worth investigating further. There could be bias w.r.t. the age of the person that applies for credit.

Adding background knowledge to explanations has the potential to significantly improve their quality [2,46]. Ignoring it can lead to explanations that disregard the needs of the user, or do not fit the reality of our world - depending on its purpose. An example is the minimum credit amount ("a credit of at least 10,000 EUR"). Further, interactivity arises naturally in reasonx: the user can flexibly query it, choosing queries that best fit to her questions, e.g., by adding constraints, and thereby building an own, personalized explanation.

Here, we focus on the CLP layer of reasonx. The Python layer and case studies at the user level are thoroughly presented in a companion paper [47].

The paper is structured as follows. In Sect. 2, we discuss background and related work. Section 3 describes the syntax, semantics, and meta-programming features of CLP that reasonx builds on. The architecture of reasonx is described in Sect. 4. We summarize contributions and future work in Sect. 5.

2 Background and Related Work

Logic and Knowledge in XAI. Several XAI approaches have used (propositional) logic rules as forms of model-agnostic explanations both locally [17,28,36] and globally [41]. Such approaches, however, do not allow for reasoning over produced explanations. Surveys on work at the intersection between symbolic and sub-symbolic methods (incl. argumentation and abduction) are [10,14,20].

Contrastive Explanations. Contrastive explanations[1] (CEs), i.e., instances similar to those to explain but with different labels assigned by the black-box (BB) classifier, are a key element in causal approaches to interpretability [11, 48]. [49] introduces contrastive explanations to the field of XAI, with several extensions [21,39]. Moreover, while [9] argues in favor of CEs from a psychological point of view, [27,30] make clear that explanations in a contrastive form are highly desirable for (lay) end-users.

Interactivity. Interactivity aligns closely with our working definition of an explanation: "[...] an interaction, or an exchange of information", where it crucially matters to *whom* the explanation is given, and for *what* purpose [46]. [45] convincingly arguments for interactivity by presenting the glass-box tool [43]. [25] confirms the relevance of interactivity via an interview study with practitioners.

Explanations and Decision Trees. Closely linked work is presented by a series of papers of Sokol et al., introducing explanations for DTs [42], generalizing it to local surrogate models [44], and exploiting interactivity [43]. Again, the main difference to our work is our reliance on CLP, and thus reasoning capabilities. Another related work is [4], providing CEs via (actual) causality.

Embedding Decision Trees into Constraints. In this paper, we assume that the DT is already available. We reason over the DT by encoding it as a set of linear constraints. This problem, known as embedding [6], requires to satisfy $c(x, y) \Leftrightarrow f(x) = y$, where $f(x)$ is the class as predicted by the DT, x the input vector consisting of discrete and/or continuous variables, and c is a constraint of some type. We adopt a rule-based encoding, which takes space in $O(N \log N)$ where N is the number of nodes in the DT. Other encodings, such as Table and MDD [6], require discretization of continuous features, thus losing the power of reasoning over linear constraints over reals.

3 Preliminaries: Constraint Logic Programming

Logic programming (LP) is a declarative approach to problem-solving based on logic rules in the form of Horn clauses [1]. It supports reasoning under various settings, e.g., deductive, inductive, abductive, and meta-reasoning [13,40]. Starting with Prolog [12], LP has been extended in several directions, as per expressivity and efficiency [24]. Constraint logic programming (CLP) augments logic programming with the ability to solve constrained problems [19]. The CLP

[1] To avoid confusion with the concept of counterfactuals as understood in the statistical causality literature, and following [27], we use the term contrastive explanations.

scheme defines a family of languages, CLP($\mathcal{C}$), that is parametric in the constraint domain $\mathcal{C}$. We are interested in CLP($\mathcal{R}$), namely the constraint domain over the reals. We use the SWI Prolog system [50] implementation.

We rely on meta-programming, a powerful technique that allows a LP to manipulate programs encoded as terms. This is extended in CLP by encoding constraints as terms.

Further, CLP($\mathcal{R}$) offers mixed integer linear programming (MILP) optimization functionalities [26]. Common predicates include the calculation of the supremum and the infimum of an expression w.r.t. the solutions of the constraint store. Complex constraint meta-reasoning procedures are based on such predicates, some examples are [3,38].

4 Explaining via Reasoning: REASONX

REASONX consists of two layers. The top layer in Python is designed for integration with the `pandas` and `scikit-learn` standard libraries for data storage and model construction. Meta-data, models, and user constraints specified at this level are parsed and transformed into Prolog facts. The bottom layer is in CLP($\mathcal{R}$) and it is written in SWI Prolog [50].

REASONX relies on a DT, the *base model*. Such a tree can be: (a) the model to be explained/reasoned about[2]; (b) a global surrogate of an opaque ML model; (c) a local surrogate trained to mimic a BB model in the neighborhood of the (local) instance to explain. In cases (b) and (c), the surrogate model is assumed to have good fidelity in reproducing the decisions of the black-box. This is reasonable for local models, i.e., in case (c). Learning the tree over a local neighborhood has been a common strategy in perturbation-based XAI methods such as LIME [35]. Following, we present an excerpt of the initialization code:

```
> r = reasonx.ReasonX(...)
> r.model(clf)
```

where the meta-data about the features are passed to the object `r` during its creation, and the DT `clf` is passed over. There can be more than one base model to account for different ML models, e.g., Neural Networks vs ensembles. The user can declare and reason about one or more instances, factual or contrastive, by specifying their class value. Each instance refers to a specific base model. The instance does not need to be fully specified, as in existing XAI methods. For example, an instance `F` can be declared with only the following characteristics:

```
> r.instance('F', label=1)
> r.constraint('F.age = 30, F.capitalloss >= 1000')
```

to intensionally denote a persons with age of 30 and capital loss of at least 1,000. Background knowledge can be expressed through linear constraints over features of instances. E.g., by declaring another instance `CE` classified differently by the

[2] While DTs are generally thought interpretable, it depends on their size/depth. Large DTs are hard to reason about, especially in a contrastive explanation scenario.

base model (the contrastive instance), the following constraints require that the contrastive instance must not be younger, and has a larger capital loss:

```
> r.instance(‘CE’, label=0)
> r.constraint(‘F.age <= CE.age, CE.capitalloss >= F.capitalloss + 500’)
```

The output of REASONX consists of constraints for which the declared instances are classified as expected by the DT(s) and such that user and implicit constraints on feature data types are satisfied. The output can be projected on only some of the instances or of the features:

```
> r.solveopt(project=[‘CE’])
> ---
> Answer: 30 <= CE.age, F.capitalloss >= 1500, CE.hoursperweek >= 40.0
```

where `30 <= CE.age, F.capitalloss >= 1500` are entailed by the constraints and `CE.hoursperweek >= 40.0` is due to conditions in the DT. Moreover, the user can specify a distance function for the minimization problem to derive the closest contrastive example, e.g., as in `solveopt(minimize=‘l1norm(F, CE)’)`.

4.1 Embeddings into CLP

We are agnostic about the learning algorithm that produces the base model(s). Features can be nominal, ordinal, or continuous. Ordinal features are coded as consecutive integer values (some preprocessing is offered in REASONX). Nominal features can be one-hot encoded or not. When embedding the DT into CLP, we force one-hot encoding of nominal features anyway, and silently decode back when returning the answer constraints to the user. A nominal feature x_i is one-hot encoded into $x_i^{v_1}, \ldots, x_i^{v_k}$ with $v_1, \ldots, v_k$ being the distinct values in the domain of x_i. We assume that the split conditions from a parent node to a child node are of the form $\mathbf{a}^T\mathbf{x} \simeq b$, where $\mathbf{x}$ is the vector of features x_i's. The following common split conditions are covered by such an assumption:

- axis-parallel splits for continuous and ordinal features, i.e., $x_i \leq b$ or $x_i > b$;
- linear splits for continuous features: $\mathbf{a}^T\mathbf{x} \leq b$ or $\mathbf{a}^T\mathbf{x} > b$;
- (in)equality splits for nominal features: $x_i = v$ or $x_i \neq v$; in terms of one-hot encoding, they respectively translate into $x_i^v = 1$ or $x_i^v = 0$.

Axis parallel and equality splits are used in CART [7] and C4.5 [34]. Linear splits are used in oblique [32] and optimal decision trees [5]. Linear model trees combine axis parallel splits at nodes and linear splits at leaves [16].

Embedding Base Model(s) into Prolog Facts. Each path (root to the leaf in the DT), is translated into a fact, a conjunction of linear split conditions:

$$\texttt{path}(m, [\mathbf{x}], [\mathbf{a}_1^{\mathrm{T}}\mathbf{x} \simeq b_1, \ldots, \mathbf{a}_{\mathrm{k}}^{\mathrm{T}}\mathbf{x} \simeq b_k], c, p).$$

where m is an id of the decision tree, $[\mathbf{x}]$ a list of (Prolog) variables representing the features, c the class predicted at the leaf, p the confidence of the prediction, and $[\mathbf{a}_1^T\mathbf{x} \simeq b_1, \ldots, \mathbf{a}_k^T\mathbf{x} \simeq b_k]$ the list of the k split conditions.

Encoding Instances. Each instance is represented by a list of Prolog variables. The mapping between names and variables is positional, and decoding is stored in a predicate `feature(`i, *varname*`)` where i is a natural number and *varname* a constant symbol, e.g., `vAge`. All instances are collectively represented by a list of lists of variables *vars*. Further, REASONX is defining a utility predicate `data_instance` with instance's meta-data.

Encoding Implicit Constraints (Ψ). Constraints on the features $\mathbf{x}$ of each instance derive from their data types. We call them "implicit" because the system can generate them from meta-data:

- for continuous features: $x_i \in \mathcal{R}$;
- for ordinal features: $x_i \in \mathcal{Z}$ and $m_i \leq x_i \leq M_i$ where $dom(x_i) = \{m_i, \ldots, M_i\}$;
- for one-hot encoded nominal features: $x_i^{v_1}, \ldots, x_i^{v_k} \in \mathcal{Z}$ and $\wedge_{j=1}^{k} 0 \leq x_i^{v_j} \leq 1$ and $\sum_{j=1}^{k} x_i^{v_j} = 1$;

Constraints for ordinal and nominal features are computed by the Prolog predicates `ord_constraints(`*vars*, `COrd)` and `cat_constraints(`*vars*`, CCat)` respectively. We denote by Ψ the conjunction of all implicit constraints.

Encoding User Constraints (Φ). The following background knowledge, loosely categorized as in [23], can be readily expressed in REASONX:

Feasibility. Constraints concerning the possibility of feature changes, and how these depend on previous values or (changes of) other features:
- *Immutability*: a feature cannot/must not change.
- *Mutable but not actionable*: the change is only a result of changes in features it depends upon.
- *Actionable but constrained*: the feature can be changed only under some condition.

Consistency. Constraints aiming at specific domain values a feature can take.

Constraints specified in Python are parsed and transformed into a list of CLP constraints. An interpreter of expressions is provided which returns a list of linear constraints over variables. The only non-linear constraint is equality of nominal values and is translated exploiting one-hot-encoding of nominal features.

Encoding Distance Functions. We simplify the optimization proposed in [49] by the assumption that declared instances have a class label[3]. The distance function is defined as a linear combination of L_1 and L_∞ norms for ordinal and continuous features and of a simple matching distance for nominal features:

$$\min \sum_{i \text{ nominal}} \mathbb{1}(x_{cf,i} \neq x_{f,i}) + \beta \sum_{i \text{ ord., cont.}} |x_{cf,i} - x_{f,i}| + \gamma \max_{i \text{ ord., cont.}} |x_{cf,i} - x_{f,i}| \quad (1)$$

[3] The split conditions from root to leaf do not necessarily lead to the same class label with 100% probability. REASONX includes a parameter in the declaration of an instance to require a minimum confidence value of the required class.

where β and γ denote parameters. L_1 and L_∞ norms are calculated over max-min normalized values to account for different units of measures. See [22,49] for a discussion. To solve the MILP problem, we need to linearize the minimization. This leads to additional constraints and slack variables.

4.2 The Core Meta-Interpreter of REASONX

We reason on constraints as theories and design operators for composing such theories. The core engine of REASONX is implemented as a Prolog meta-interpreter of expressions over those operators.

A (logic) theory is a set of formulas, from which one is interested to derive implied formulas, and a logic program is itself a theory [8]. In our context, a theory consists of a set of linear constraints $\{c_i\}_i$ to be interpreted as the disjunction $\vee_i \ c_i$. Theories are coded in LP by exploiting its non-deterministic computational model, i.e., each c_i's is returned by a clause in the program. The language of expressions over theories is closed: operators map one or more theories into a theory. The following theories are included:

`typec` the theory with only the conjunction $\wedge_{c \in \Psi} \ c$ of the implicit constraints;
`userc` the theory with only the conjunction $\wedge_{c \in \Phi} \ c$ of the user constraints;
`inst(I)` the theory of constraints $\wedge_i \ \mathbf{a}_i^T \mathbf{x} \simeq b_i$ where $\mathbf{x}$ are features of the instance `I`, and primitive constraints $\mathbf{a}_i^T \mathbf{x} \simeq b_i$ are those in the path of the decision tree `M` the instance refers to.

We provide the following operators on theories: the cross-product of constraints of theories, the subset of constraints in a theory that are satisfiable, the projection of constraints in a theory over a set of variables, and the subset of constraints in a theory that minimize a certain (distance) function.

The queries to the CLP layer of REASONX can be answered by a Prolog query over the predicates `instvar` (building *vars*), `proj_vars` (computing which of those variables are to be projected in the output), and `solve` (evaluating expressions over the cross-product of `typec`, `userc`, and the theories `inst(I)` for all instances `I`).

5 Conclusion

We presented REASONX, a declarative XAI tool that relies on linear constraint reasoning, solving for background knowledge, and for interaction with the user at a high abstraction and intensional level. These features make it a unique tool when compared to instance-level approaches commonly adopted for explaining ML models. We aim at extending REASONX along three directions: *i)* the implementation of additional constraints, possibly with non-linear solvers, *ii)* an extensive evaluation based on some theoretical measures, as well as through user-studies [37] and real-world data, and *iii)* extension to non-structured data, such as images and text, e.g., through the integration of concepts [15].

Software. REASONX is released open source at https://github.com/lstate/REASONX.

Acknowledgments. Work supported by the European Union's Horizon 2020 research and innovation programme under Marie Sklodowska-Curie Actions for the project NoBIAS (g.a. No. 860630), and by the NextGenerationEU program within the PNRR-PE-AI scheme (M4C2, investment 1.3, line on Artificial Intelligence) for the project FAIR (Future Artificial Intelligence Research). This work reflects only the authors' views and the European Research Executive Agency (REA) is not responsible for any use that may be made of the information it contains.

References

1. Apt, K.: From Logic Programming to Prolog. Prentice Hall, London New York (1997)
2. Beckh, K., et al.: Explainable machine learning with prior knowledge: An overview. CoRR abs/2105.10172 (2021)
3. Benoy, F., King, A., Mesnard, F.: Computing convex hulls with a linear solver. Theory Pract. Log. Program. **5**(1–2), 259–271 (2005)
4. Bertossi, L.E.: Declarative approaches to counterfactual explanations for classification. CoRR abs/2011.07423 (2020)
5. Bertsimas, D., Dunn, J.: Optimal classification trees. Mach. Learn. **106**(7), 1039–1082 (2017). https://doi.org/10.1007/s10994-017-5633-9
6. Bonfietti, A., Lombardi, M., Milano, M.: Embedding decision trees and random forests in constraint programming. In: Michel, L. (ed.) CPAIOR 2015. LNCS, vol. 9075, pp. 74–90. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18008-3_6
7. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth (1984)
8. Brogi, A., Mancarella, P., Pedreschi, D., Turini, F.: Theory construction in computational logic. In: Jacquet, J. (ed.) Constructing Logic Programs, pp. 241–250. Wiley (1993)
9. Byrne, R.M.J.: Counterfactuals in explainable artificial intelligence (XAI): evidence from human reasoning. In: IJCAI, pp. 6276–6282. ijcai.org (2019)
10. Calegari, R., Ciatto, G., Omicini, A.: On the integration of symbolic and sub-symbolic techniques for XAI: a survey. Intelligenza Artificiale **14**(1), 7–32 (2020)
11. Chou, Y., Moreira, C., Bruza, P., Ouyang, C., Jorge, J.A.: Counterfactuals and causability in explainable artificial intelligence: theory, algorithms, and applications. Inf. Fusion **81**, 59–83 (2022)
12. Clocksin, W.F., Mellish, C.S.: Programming in Prolog. Using the ISO Standard. Springer, Heidelberg (2003)
13. Cropper, A., Dumancic, S.: Inductive logic programming at 30: a new introduction. J. Artif. Intell. Res. **74**, 765–850 (2022)
14. Dietz, E., Kakas, A.C., Michael, L.: Argumentation: a calculus for human-centric AI. Front. Artif. Intell. **5**, 955579 (2022)
15. Donadello, I., Dragoni, M.: SeXAI: introducing concepts into black boxes for explainable Artificial Intelligence. In: XAI.it@AI*IA. CEUR Workshop Proceedings, vol. 2742, pp. 41–54. CEUR-WS.org (2020)
16. Frank, E., Wang, Y., Inglis, S., Holmes, G., Witten, I.H.: Using model trees for classification. Mach. Learn. **32**(1), 63–76 (1998)
17. Guidotti, R., Monreale, A., Giannotti, F., Pedreschi, D., Ruggieri, S., Turini, F.: Factual and counterfactual explanations for black box decision making. IEEE Intell. Syst. **34**(6), 14–23 (2019)

18. 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)
19. Jaffar, J., Michaylov, S., Stuckey, P.J., Yap, R.H.C.: The CLP(R) language and system. ACM Trans. Program. Lang. Syst. **14**(3), 339–395 (1992)
20. Kakas, A.C., Michael, L.: Abduction and argumentation for explainable machine learning: a position survey. CoRR abs/2010.12896 (2020)
21. Kanamori, K., Takagi, T., Kobayashi, K., Arimura, H.: DACE: distribution-aware counterfactual explanation by mixed-integer linear optimization. In: IJCAI, pp. 2855–2862. ijcai.org (2020)
22. Karimi, A., Barthe, G., Balle, B., Valera, I.: Model-agnostic counterfactual explanations for consequential decisions. In: AISTATS. Proceedings of Machine Learning Research, vol. 108, pp. 895–905. PMLR (2020)
23. Karimi, A., Barthe, G., Schölkopf, B., Valera, I.: A survey of algorithmic recourse: definitions, formulations, solutions, and prospects. CoRR abs/2010.04050 (2020)
24. Körner, P., et al.: Fifty years of Prolog and beyond. Theory Pract. Log. Program. **22**(6), 776–858 (2022)
25. Lakkaraju, H., Slack, D., Chen, Y., Tan, C., Singh, S.: Rethinking explainability as a dialogue: a practitioner's perspective. CoRR abs/2202.01875 (2022)
26. Magatão, L.: Mixed integer linear programming and constraint logic programming: towards a unified modeling framework. Ph.D. thesis, Federal University of Technology - Paraná, Brazil (2010)
27. Miller, T.: Explanation in artificial intelligence: insights from the social sciences. Artif. Intell. **267**, 1–38 (2019)
28. Ming, Y., Qu, H., Bertini, E.: Rulematrix: Visualizing and understanding classifiers with rules. IEEE Trans. Vis. Comput. Graph. **25**(1), 342–352 (2019)
29. Minh, D., Wang, H.X., Li, Y.F., Nguyen, T.N.: Explainable artificial intelligence: a comprehensive review. Artif. Intell. Rev. **55**(5), 3503–3568 (2022)
30. Mittelstadt, B.D., Russell, C., Wachter, S.: Explaining explanations in AI. In: FAT, pp. 279–288. ACM (2019)
31. Molnar, C.: Interpretable Machine Learning. A Guide for Making Black Box Models Explainable (2019). https://christophm.github.io/interpretable-ml-book
32. Murthy, S.K., Kasif, S., Salzberg, S.: A system for induction of oblique decision trees. J. Artif. Intell. Res. **2**, 1–32 (1994)
33. Ntoutsi, E., et al.: Bias in data-driven artificial intelligence systems - an introductory survey. WIREs Data Min. Knowl. Discov. **10**(3), e1356 (2020)
34. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, Burlington (1993)
35. Ribeiro, M.T., Singh, S., Guestrin, C.: "Why should I trust you?": Explaining the predictions of any classifier. In: KDD, pp. 1135–1144. ACM (2016)
36. Ribeiro, M.T., Singh, S., Guestrin, C.: Anchors: high-precision model-agnostic explanations. In: AAAI, pp. 1527–1535. AAAI Press (2018)
37. Rong, Y., Leemann, T., Nguyen, T., Fiedler, L., Seidel, T., Kasneci, G., Kasneci, E.: Towards human-centered explainable AI: user studies for model explanations. CoRR abs/2210.11584 (2022)
38. Ruggieri, S.: Deciding membership in a class of polyhedra. In: ECAI. Frontiers in Artificial Intelligence and Applications, vol. 242, pp. 702–707. IOS Press (2012)
39. Russell, C.: Efficient search for diverse coherent explanations. In: FAT, pp. 20–28. ACM (2019)
40. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Pearson Education, London (2003)

41. Setzu, M., Guidotti, R., Monreale, A., Turini, F., Pedreschi, D., Giannotti, F.: Glocalx - from local to global explanations of black box AI models. Artif. Intell. **294**, 103457 (2021)
42. Sokol, K.: Towards Intelligible and Robust Surrogate Explainers: A Decision Tree Perspective. Ph.D. thesis, School of Computer Science, Electrical and Electronic Engineering, and Engineering Maths, University of Bristol (2021)
43. Sokol, K., Flach, P.A.: Glass-box: explaining AI decisions with counterfactual statements through conversation with a voice-enabled virtual assistant. In: IJCAI, pp. 5868–5870. ijcai.org (2018)
44. Sokol, K., Flach, P.A.: LIMEtree: interactively customisable explanations based on local surrogate multi-output regression trees. CoRR abs/2005.01427 (2020)
45. Sokol, K., Flach, P.A.: One explanation does not fit all. Künstliche Intell. **34**(2), 235–250 (2020)
46. State, L.: Logic programming for XAI: a technical perspective. In: ICLP Workshops. CEUR Workshop Proceedings, vol. 2970. CEUR-WS.org (2021)
47. State, L., Ruggieri, S., Turini, F.: Reason to explain: interactive contrastive explanations (REASONX). CoRR abs/2305.18143 (2023)
48. Stepin, I., Alonso, J.M., Catalá, A., Pereira-Fariña, M.: A survey of contrastive and counterfactual explanation generation methods for explainable artificial intelligence. IEEE Access **9**, 11974–12001 (2021)
49. Wachter, S., et al.: Counterfactual explanations without opening the black box. Harv. JL Tech. **31**, 841 (2017)
50. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theory Pract. Log. Program **12**(1–2), 67–96 (2012)