Declarative Approaches to Counterfactual Explanations for Classification∗

LEOPOLDO BERTOSSI
Universidad Adolfo Ibáñez, Faculty of Engineering and Sciences
and
Millenium Institute on Foundations of Data (IMFD)
Santiago, Chile.
leopoldo.bertossi@uai.cl

submitted xxx; revised xxx; accepted xxx

Abstract

We propose answer-set programs that specify and compute counterfactual interventions on entities that are input on a classification model. In relation to the outcome of the model, the resulting counterfactual entities serve as a basis for the definition and computation of causality-based explanation scores for the feature values in the entity under classification, namely *responsibility scores*. The approach and the programs can be applied with black-box models, and also with models that can be specified as logic programs, such as rule-based classifiers. The main focus of this work is on the specification and computation of *best* counterfactual entities, i.e. those that lead to maximum responsibility scores. From them one can read off the explanations as maximum responsibility feature values in the original entity. We also extend the programs to bring into the picture semantic or domain knowledge. We show how the approach could be extended by means of probabilistic methods, and how the underlying probability distributions could be modified through the use of constraints.

KEYWORDS: Classification, explanations, counterfactuals, causality, answer-set programming, constraints

1 Introduction

Providing explanations to results obtained from machine-learning (ML) models has been recognized as critical in many applications. It has also become an active research direction in explainable ML (Molnar 2020), and the broader area of *explainable AI*. Explanations become particularly relevant when decisions are automatically made by those models, possibly with serious consequences for stake holders. Since most of those models are algorithms learned from training data, providing explanations may not be easy or possible. These models are or can be seen as *black-box models*. Even if the components of the models, say their structure, mathematical functions, and parameters, are relatively clear and accessible, characterizing

∗ In memory of Prof. Jack Minker (1927-2021), a scientist, a scholar, a visionary, a generous, wise and committed man.

and measuring the relevance of a particular feature value for the classification of an entity may not be that clear from a sheer inspection of the model.

In AI, explanations have been investigated, among other areas, under *actual causality* (Halpern and Pearl 2005), where *counterfactual interventions* on a causal *structural model* are central. They are hypothetical updates on the model's variables, to explore if and how the outcome of the model changes or not. In this way, explanations for an original output are defined and computed. Counterfactual interventions have been used, explicitly or implicitly, with ML models, in particular with classification models (Martens and Provost 2014; Wachter et al. 2017; Russell 2019; Karimi et al. 2020a; Datta et al. 2016; Bertossi et al. 2020). Counterfactual explanations for query answers from databases have also been investigated (Meliou et al. 2010; Bertossi and Salimi 2017a; Bertossi 2020b), an area that is not unrelated to classification results (c.f. Section 4).

In this work we start by introducing and formalizing the notion of *counterfactual explanation* for the classification of an entity $\mathbf{e}$ (think of a finite record of feature values) with a certain label ℓ. Counterfactual explanations are alternative versions, $\mathbf{e}'$, of $\mathbf{e}$ that differ from $\mathbf{e}$ by a set of feature values, but are classified with a different label ℓ'. In this work, "best counterfactual explanations" come in two forms, as s-explanations and c-explanations. The former minimize (under set inclusion) the *set* of features whose values are changed; while the latter, minimize the *number* of changed feature values.

Next, we use counterfactual explanations to define the x-Resp score, an explanation *responsibility score* for a feature value of the entity under classification. The idea is to identify and compute the *most responsible* feature values for the outcome. This score is adapted from the general notion of responsibility used in actual causality (Chockler and Halpern 2004).

More specifically, in this work, we concentrate our interest mostly and mainly on specifying and computing, for a given and fixed classified entity $\mathbf{e}$, *all the best explanations* as represented as counterfactual entities (that have feature values changed and a different label), where "best" for a counterfactual entity means that the number of changes of feature values (with respect to $\mathbf{e}$) takes a minimum value (the c-explanations mentioned above). At the same time, we are also interested in the *maximum-responsibility feature values*. These two goals are directly related: The maximum-responsibility feature values (in the original entity $\mathbf{e}$) are exactly those that appear as changed in a best counterfactual entity.

The x-Resp score belongs to the family of *feature attribution* scores, among which one finds the popular Shap score (Lundberg et al. 2020). While Shap is based on the Shapley value that is widely used in game theory, the x-Resp score is based on actual causality. Experimental results with an extended version (that we briefly describe) of the responsibility score investigated in this work, and comparisons with other scores, in particular, with Shap, are reported in (Bertossi et al. 2020). However, only a fully procedural approach to the x-Resp score was followed in (Bertossi et al. 2020). Furthermore, for performance and comparison-related reasons, the number of counterfactually changeable feature values was bounded a priori.

Answer-set programming (ASP) is an elegant and powerful logic programming

paradigm that allows for declarative specifications of a domain or a problem. From those specifications one can do logical reasoning, and quite usefully, non-monotonic reasoning (Brewka et al. 2011; Gelfond and Kahl 2014). ASP has been applied to the specification and solution of hard combinatorial problems, and also to different tasks related to databases. Among the latter, we find the specification of repairs of databases w.r.t. integrity constraints, virtual data integration, specification of privacy views (Bertossi 2011; Bertossi 2019), and causality in databases (Bertossi 2020b).

In this work we also introduce *Counterfactual Intervention Programs* (CIPs), which are answer-set programs (ASPs) that specify counterfactual interventions and counterfactual explanations, and also allow to specify and compute the responsibility scores. More specifically, CIPs are designed to be used to compute: (a) counterfactual explanations (best or not), (b) best counterfactual explanations (i.e. c-explanations), and (c) highest-responsibility features values (in the original entity), that is, with maximum x-Resp score.

As already mentioned above, c-explanations lead to maximum responsibility scores, and maximum-responsibility scores are associated to c-explanations. However, in order to compute a maximum-responsibility score (or a feature value which attains it), we may not need all the best counterfactual entities that "contain" that particular feature value. Actually, only one of them would be good enough. Still, our CIPs compute all the counterfactual entities, and all (and only) the best counterfactual entities if so required.

In this regard, it is worth mentioning that *counterfactual explanations*, best or not, are interesting *per se*, in that they could show what could be done differently in order to be obtain a different classification (Schleich et al. 2021). For example, someone, represented as an entity $\mathbf{e}$, who applies for a loan at a bank, and is deemed by the bank's classifier as unworthy of the loan, could benefit from knowing that a reduction of the number of credit cards he/she owns might lead to a positive assessment. This counterfactual explanation would be considered as *actionable* (or *feasible* or a *recourse*) (Ustun et al. 2019; Karimi et al. 2020b).

Furthermore, having all (or only the best) counterfactual explanations, we could think of doing different kinds of meta-analysis and meta-analytics on top of them. In some cases, query answering on the CIP can be leveraged, e.g. to know if a highest responsibility feature value appears as changed in every best counterfactual entity (c.f. Section 9).

CIPs can be applied to black-box models that can be invoked from the program; and also with any classifier that can be specified within the program, such as a rule-based classifier. Decision trees for classification can be expressed by means of rules. As recent research shows, other established classifiers, such as random forests, Bayesian networks, and neural networks, can be compiled into Boolean circuits (Shi et al. 2020; Shih et al. 2018; Choi et al. 2020; Narodytska et al. 2010), opening the possibility of specifying them by means of ASPs.

As we will show in this work, our declarative approach to counterfactual interventions is particularly appropriate for bringing into the game additional, declarative, semantic knowledge. As we show, we could easily adopt *constraints* when comput-

ing counterfactual entities and scores. In particular, these constraints can be used to concentrate only on *actionable* counterfactuals (as defined by the constraint). In the loan example, we could omit counterfactual entities or explanations that require from the applicant to reduce his/her age. Imposing constraints in combination with purely procedural approaches is much more complicated. In our case, we can easily and seamlessly integrate logic-based semantic specifications with declarative programs, and use the generic and optimized solvers behind ASP implementations. In this work we include several examples of CIPs specified in the language of the *DLV* system (Leone et al. 2006) and its extensions, and run on them.

We establish that ASPs are the right declarative and computational tool for our problem since they can be used to: (a) Compute all the counterfactual explanations, in particular, the best ones. (b) Provide through each model of the program all the information about a particular counterfactual explanation. (c) Compute maximum-responsibility scores and the counterfactual explanations associated to them. (d) In many cases, specify as a single program the classifier together with the explanation machinery. (d) Alternatively, call a classifier from the program as an external predicate defined in, say Python. (e) Bring into the picture additional, logic-based knowledge, such as semantic and actionability constraints. (f) Do query answering, under the skeptical and brave semantics, to analyze at a higher level the outcomes of the explanation process. Furthermore, and quite importantly, the ASPs we use match the intrinsic data complexity of the problem of computing maximum-responsibility feature values, as we will establish in this work.

We discuss why and how the responsibility score introduced and investigated early in this work has to be extended in some situations, leading to a generalized probabilistic responsibility score introduced in (Bertossi et al. 2020). After doing this, we show how logical constraints that capture domain knowledge can be used to modify the underlying probability distribution on which a score is defined.

This paper is structured as follows. In Section 2 we provide background material on classification models and answer-set programs. In Section 3 we introduce counterfactual interventions and the responsibility score. In Section 4, we investigate the data complexity of the computation of maximum-responsibility feature values. In Section 5 we introduce and analyze the CIPs. In Section 6, we consider some extensions of CIPs that can be used to capture additional domain knowledge. In Section 7, and for completeness, we briefly motivate and describe the probabilistic responsibility score previously introduced in (Bertossi et al. 2020). In this regard, we also show how to bring logical constraints into the probabilistic setting of the responsibility score. In Section 8, the discuss related work. In Section 9, we conclude with a discussion and concluding remarks.

This paper is an extended and invited submission associated to the conference paper (Bertossi 2020a). The current version, in addition to improving and making more precise the presentation in comparison with the conference version, includes a considerable amount of new material. Section 2, containing background material, is new. In particular, it contains a new example of a decision-tree classifier that is used as an additional running example throughout the paper. Section 4 on the complexity of x-Resp computation is completely new, so as Section 5.2 that retakes

complexity at the light of CIPs. This version also contains several examples of the use of the *DLV* system for specifying and running CIPs. There were no such examples in the conference version. Section 7 is new. It contains two subsections; the first dealing with non-binary features and the need to extend in probabilistic terms the definition of the x-Resp score; and the second, with the combination of probabilistic uncertainty and domain knowledge. We now have the new Section 8 on related work. The final Section 9 has been considerably revised and extended.

2 Background

2.1 Classification models and counterfactuals

In a particular domain we may have a finite set $\mathcal{F} = \{F_1, \ldots, F_n\}$ of *features* (a.k.a. as *variables*). More precisely, the features F_i are functions that take values in their *domains* $Dom(F_i)$.[1] As is commonly the case, we will assume that these domains are *categorical*, i.e. finite and without any a priori order structure. These features are used to describe or represent *entities* in an *underlying population* (or application domain), as records (or tuples) $\mathbf{e}$ formed by the values the features take on the entity. Actually, with a bit of abuse of notation, we represent entities directly as these records of feature values: $\mathbf{e} = \langle F_1(\mathbf{e}), \ldots, F_n(\mathbf{e})\rangle$. A feature F is said to be *Boolean* or *propositional* if $Dom(F) = \{\mathsf{true}, \mathsf{false}\}$, for which we sometimes use $\{\mathsf{yes}, \mathsf{no}\}$ or simply, $\{1, 0\}$.

For example, if the underlying population contains people, and then, each entity represents a person, features could be $\mathcal{F} = \{\mathsf{Weight}, \mathsf{Age}, \mathsf{Marrid}, \mathsf{EdLevel}\}$, with domains: $Dom(\mathsf{Weight}) = \{\mathsf{overweight}, \mathsf{chubby}, \mathsf{fit}, \mathsf{slim}, \mathsf{underweight}\}$; $Dom(\mathsf{Age}) = \{\mathsf{old}, \mathsf{middle}, \mathsf{young}, \mathsf{child}\}$; $Dom(\mathsf{Married}) = \{\mathsf{true}, \mathsf{false}\}$; $Dom(\mathsf{EdLevel}) = \{\mathsf{low}, \mathsf{medium}, \mathsf{high}, \mathsf{top}\}$. A particular entity could be (represented by) $\mathbf{e} = \langle \mathsf{fit}, \mathsf{young}, \mathsf{false}, \mathsf{top}\rangle$. Here, Married is a propositional (or binary) feature.

A *classifier*, $\mathcal{C}$, for a domain of entities $\mathcal{E}$ is a computable function $\mathcal{C} : \mathcal{E} \to L$, where $L = \{\ell_1, \ldots, \ell_k\}$ is a set of *labels*. We classify the entity by assigning a label to it. We do not have to assume any order structure on L. In the example above, we may want to classify entities in the people's population according to how appropriate they are for military service. The set of labels could be $L = \{\mathsf{good}, \mathsf{maybe}, \mathsf{noway}\}$, then the classifier could be such, that $\mathcal{C}(\langle \mathsf{fit}, \mathsf{young}, \mathsf{false}, \mathsf{top}\rangle) = \mathsf{good}$, but $\mathcal{C}(\langle \mathsf{overweight}, \mathsf{old}, \mathsf{true}, \mathsf{low}\rangle) = \mathsf{noway}$. When the set of labels has two values, typically, $\{\mathsf{yes}, \mathsf{no}\}$, we have a *binary or Boolean classifier*. In this work we will consider mostly binary classifiers.

Example 2.1
This is a popular example taken from (Mitchell 1997). Consider the set of features $\mathcal{F} = \{\mathsf{Outlook}, \mathsf{Humidity}, \mathsf{Wind}\}$, with $Dom(\mathsf{Outlook}) = \{\mathsf{sunny}, \mathsf{overcast}, \mathsf{rain}\}$, $Dom(\mathsf{Humidity}) = \{\mathsf{high}, \mathsf{normal}\}$, $Dom(\mathsf{Wind}) = \{\mathsf{strong}, \mathsf{weak}\}$. An entity under classification has a value for each of the features, e.g. $\mathbf{e} = ent(\mathsf{sunny}, \mathsf{normal}, \mathsf{weak})$.

[1] This is customary parlance, but, since a feature is a function, in Mathematics we would call this *the range* of F_i.

The problem is about deciding about playing tennis or not under the conditions represented by that entity, which can be captured as a classification problem, with labels yes or no (sometimes we will use 1 and 0, resp.).

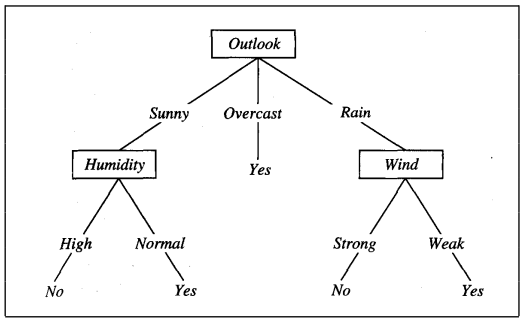

Fig. 1. A Decision Tree

The Boolean classifier is given as a decision-tree, as shown in Figure 1. The decision is computed by following the feature values along the branches of the tree. The entity $\mathbf{e}$ at hand gets label yes. □

Of course, a decision tree could be just arbitrarily given, but the common situation is *to learn* it from training examples, that is from a collection of entities that come already with an assigned label. In more general terms, building a classifier, $\mathcal{C}$, from a set of training data, i.e. a set of pairs $T = \{\langle \mathbf{e}_1, \ell(\mathbf{e}_1)\rangle, \ldots, \langle \mathbf{e}_M, \ell(\mathbf{e}_M)\rangle\}$, with $\ell(\mathbf{e}_i) \in L$, is about learning a (computational) model for the label function L for the entire domain of entities, beyond T. We say that L "represents" the classifier $\mathcal{C}$. This *supervised learning* task is one of the most common in machine learning. (C.f. (Mitchell 1997) or (Flach 2012) for great introductions to the subject.) Classifiers may take many different internal forms. They could be decision trees, random forests, rule-based classifiers, logistic regression models, neural network-based classifiers, etc.

In this work we are *not* concerned with learning classifiers. We will assume the classifier is given as an input/output computable relation, or that we know how to specify it, for example, trough a logic program. By the same token, we are *not* dealing here with any kind of program learning.

Some classes of classifiers are more "opaque" than others, i.e. with a more complex and less interpretable internal structure and results (Molnar 2020). Hence, the need for explaining the classification outcomes. The decision tree above is clearly a computable function. Since we have the classifier at hand with an explicit and clear specification, we would consider this classifier to be *interpretable*. In this direction, if we obtain a label for a particular entity at hand, we can inspect the model and explain *why* we obtained that label, and we could identify the feature values that were relevant for the outcome.

Instead of a decision tree as above, we could have, for example, a very complex

neural network. Such a model would be much more difficult to interpret, or use to explain why we got a particular output for a particular input. This kind of models are considered to be *black-box* models (or al least, *opaque*) (Rudin 2019).

Assume for a moment that we do not have the explicit classifier in Example 2.1, but we interact only with the box that contains it. We input entity $\mathbf{e} = ent(\mathsf{sunny}, \mathsf{normal}, \mathsf{weak})$, and we obtain the output yes. We want to know what are the feature values in $\mathbf{e}$ that influenced the outcome the most. Actually, we want to get a *numerical score* for each of the entity's feature values, in such a way that the higher the score, the more relevant is the feature value for the outcome.

Different scores may be defined. A popular one is Shap (Lundberg et al. 2020), which has been investigated in detail for some classes of classifiers (Arenas et al. 2021; Van den Broeck et al. 2021). In this work we concentrate on x-Resp, a *responsibility score* that was introduced, in a somewhat *ad hoc* manner, in (Bertossi et al. 2020). In this work, we present x-Resp in a causal setting, on the basis of counterfactual interventions and their responsibilities, following the general approach to causal responsibility in (Chockler and Halpern 2004).

Just for the gist, and at the light of the example at hand, we want to detect and quantify the relevance (technically, the responsibility) of a feature value in $\mathbf{e} = ent(\mathsf{sunny}, \underline{\mathsf{normal}}, \mathsf{weak})$, say for feature Humidity (underlined), by *hypothetically intervening* its value; in this case, changing it from normal to high, obtaining a new entity $\mathbf{e}' = ent(\mathsf{sunny}, \underline{\mathsf{high}}, \mathsf{weak})$, a *counterfactual version* of $\mathbf{e}$. If we input this entity into the classifier, we now obtain the label no. This is an indication that the original feature value for Humidity is indeed relevant for the original classification. Through numerical aggregations over the outcomes associated to the alternative, counterfactually intervened entities, we can define and compute the x-Resp score. The details can be found in Section 3. As mentioned in Section 1, these counterfactual entities are also interesting *per se*, and not only as a basis for the definition and computation of *feature attribution scores*.

2.2 Answer-set programming

As customary, when we talk about *answer-set programs*, we refer to *disjunctive Datalog programs with weak negation and stable model semantics* (Gelfond and Lifschitz 1991). Accordingly, an answer-set program Π consists of a finite number of rules of the form

$$A_1 \vee \ldots \vee A_n \leftarrow P_1, \ldots, P_m, not\, N_1, \ldots, not\, N_k, \tag{1}$$

where $0 \leq n, m, k$, and A_i, P_j, N_s are (positive) atoms, i.e. of the form $Q(\bar{t})$, where Q is a predicate of a fixed arity, say, ℓ, and $\bar{t}$ is a sequence of length ℓ of variables or constants. In rule (1), $A_1, \ldots, not\, N_k$ are called *literals*, with A_1 *positive*, and $not\, N_k$, *negative*. All the variables in the A_i, N_s appear among those in the P_j. The left-hand side of a rule is called the *head*, and the right-hand side, the *body*. A rule can be seen as a (partial) definition of the predicates in the head (there may be other rules with the same predicates in the head).

The constants in program Π form the (finite) Herbrand universe H of the program. The ground version of program Π, $gr(\Pi)$, is obtained by instantiating the variables in Π in all possible ways using values from H. The Herbrand base, HB, of Π contains all the atoms obtained as instantiations of predicates in Π with constants in H.

A subset M of HB is a model of Π if it satisfies $gr(\Pi)$, i.e.: For every ground rule $A_1 \vee \ldots \vee A_n \leftarrow P_1, \ldots, P_m, \, not\, N_1, \ldots, not\, N_k$ of $gr(\Pi)$, if $\{P_1, \ldots, P_m\} \subseteq M$ and $\{N_1, \ldots, N_k\} \cap M = \emptyset$, then $\{A_1, \ldots, A_n\} \cap M \neq \emptyset$. M is a minimal model of Π if it is a model of Π, and Π has no model that is properly contained in M. $MM(\Pi)$ denotes the class of minimal models of Π. Now, for $S \subseteq HB(\Pi)$, transform $gr(\Pi)$ into a new, positive program $gr(\Pi)^S$ (i.e. without *not*), as follows: Delete every rule $A_1 \vee \ldots \vee A_n \leftarrow P_1, \ldots, P_m, not\, N_1, \ldots, not\, N_k$ for which $\{N_1, \ldots, N_k\} \cap S \neq \emptyset$. Next, transform each remaining rule $A_1 \vee \ldots \vee A_n \leftarrow P_1, \ldots, P_m, \, not\, N_1, \ldots, not\, N_k$ into $A_1 \vee \ldots \vee A_n \leftarrow P_1, \ldots, P_m$. Now, S is a *stable model* of Π if $S \in MM(gr(\Pi)^S)$. Every stable model of Π is also a minimal model of Π. Stable models are also commonly called *answer sets*, and so are we going to do most of the time.

A program is *unstratified* if there is a cyclic, recursive definition of a predicate that involves negation. For example, the program consisting of the rules $a \vee b \leftarrow c, not\, d$; $d \leftarrow e$, and $e \leftarrow b$ is unstratified, because there is a negation in the mutually recursive definitions of b and e. The program in Example 2.2 below is not unstratified, i.e. it is *stratified*. A good property of stratified programs is that the models can be upwardly computed following *strata* (layers) starting from the *facts*, that is from the ground instantiations of rules with empty bodies (in which case the arrow is usually omitted). We refer the reader to (Gelfond and Kahl 2014) for more details.

Query answering under the ASPs comes in two forms. Under the *brave semantics*, a query posed to the program obtains as answers those that hold in *some* model of the program. However, under the *skeptical* (or *cautious*) semantics, only the answers that simultaneously hold in *all* the models are returned. Both are useful depending on the application at hand.

We will use disjunctive programs. However, sometimes it is possible to use instead *normal programs*, which do not have disjunctions in rule heads, and with the same stable models, in the sense that disjunctive rules can be transformed into a set of non-disjunctive rules. More precisely, the rule in (1) can be transformed into the rules:

$$\begin{aligned} A_1 \quad &\leftarrow \quad P_1, \ldots, P_m, not\, N_1, \ldots, not\, N_k, not\, A_2, \ldots, not\, A_n \\ &\ldots \\ A_n \quad &\leftarrow \quad P_1, \ldots, P_m, not\, N_1, \ldots, not\, N_k, not\, A_1, \ldots, not\, A_{n-1}. \end{aligned}$$

This *shift operation* is possible if the original program is *head-cycle free* (Ben-Eliyahu and Dechter 1994; Dantsin et al. 2001), as we define now. The *dependency graph* of a program Π, denoted $DG(\Pi)$, is a directed graph whose nodes are the atoms of the associated ground program $gr(\Pi)$. There is an arc from L_1 to L_2 iff there is a rule in $gr(\Pi)$ where L_1 appears positive in the body and L_2 appears in

the head. Π is *head-cycle free* (HCF) iff $DG(\Pi)$ has no cycle through two atoms that belong to the head of a same rule.

Example 2.2
Consider the following program Π that is already ground.

$$\begin{array}{rcl} a \vee b & \leftarrow & c \\ d & \leftarrow & b \\ a \vee b & \leftarrow & e,\ not\, f \\ e & \leftarrow & \end{array}$$

The program has two stable models: $S_1 = \{e, a\}$ and $S_2 = \{e, b, d\}$.

Each of them expresses that the atoms in it are true, and any other atom that does not belong to it, is false.

These models are incomparable under set inclusion, and they are minimal models in that any proper subset of any of them is not a model of the program.

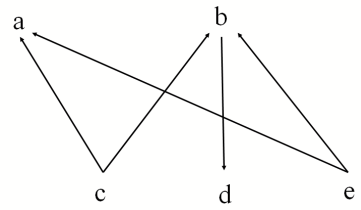

Fig. 2. Dependency graph $DG(\Pi)$

The dependency graph is shown in Figure 2. We can see that the program Π is head-cycle free, because there is no cycle involving both a and b, the atoms that appear in the disjunctive head. As a consequence, the program can be transformed into the following non-disjunctive program

$$\begin{array}{rcl} a & \leftarrow & c,\ not\, b \\ b & \leftarrow & c,\ not\, a \\ d & \leftarrow & b \end{array} \qquad \begin{array}{rcl} a & \leftarrow & e,\ not\, f,\ not\, b \\ b & \leftarrow & e,\ not\, f,\ not\, a\ . \\ e & \leftarrow & \end{array}$$

This program has the same stable models as the original one. □

We will return to HCF-programs in Section 5.2.

3 Counterfactual Explanations and the x-Resp Score

We consider *classification models*, $\mathcal{C}$, that are represented by an input/output relation. Inputs are the so-called *entities*, $\mathbf{e}$, which are represented as records, $\mathbf{e} = \langle x_1, \ldots, x_n \rangle$, where x_i is the value $F_i(\mathbf{e}) \in Dom_i := Dom(F_i)$ taken on $\mathbf{e}$ by the *feature* $F_i \in \mathcal{F} = \{F_1, \ldots, F_n\}$. The entities form a *population* $\mathcal{E}$. The output of a classifier is represented by a *label function* L that maps entities $\mathbf{e}$ to 0

or 1, the binary result of the classification. That is, to simplify the presentation, we concentrate here on binary classifiers, but this is not essential. Furthermore, we consider domains $Dom(F_i)$ with a finite number of categorical values.

In this work, we are not assuming that we have an explicit classification model, and we do not need it. All we need is to be able to invoke it. It could be a "black-box" model.

The problem is the following: Given an entity $\mathbf{e}$ that has received the label $L(\mathbf{e})$, provide an "explanation" for this outcome. In order to simplify the presentation, and without loss of generality, *we assume that label* 1 *is the one that has to be explained*. It is the "negative" outcome one has to justify, such as the rejection of a loan application.

Counterfactual explanations are defined in terms of *counterfactual interventions* that simultaneously change feature values in $\mathbf{e}$, in such a way that the updated record gets a new label. A *counterfactual explanation* for the classification of $\mathbf{e}$ is, then, an alternative entity $\mathbf{e}'$ that is classified with a label different from that of $\mathbf{e}$. In general, we are interested in counterfactual explanations that are more informative about the role of the feature values in $\mathbf{e}$, which lead to its obtained label. These are the entities that are obtained from $\mathbf{e}$ through a *minimal counterfactual interventions*. Minimality can be defined in different ways, and we adopt an abstract approach, assuming a partial order relation $\preceq$ on counterfactual interventions.

Definition 3.1
Consider a binary classifier represented by its label function L, and a fixed input entity $\mathbf{e} = \langle x_1, \ldots, x_n \rangle$, with $F_i(\mathbf{e}) = x_i$, $1 \leq i \leq n$, and $L(\mathbf{e}) = 1$.
(a) An *intervention* $\hat{\imath}$ on $\mathbf{e}$ is a set of the form $\{\langle F_{i_1}, x'_{i_1}\rangle, \ldots, \langle F_{i_K}, x'_{i_K}\rangle\}$, with $F_{i_s} \neq F_{i_\ell}$, for $s \neq \ell$, $x_{i_s} \neq x'_{i_s} \in Dom(F_{i_s})$. We denote with $\hat{\imath}(\mathbf{e})$ the record obtained by applying to $\mathbf{e}$ intervention $\hat{\imath}$, i.e. by replacing in $\mathbf{e}$ every $x_{i_s} = F_{i_s}(\mathbf{e})$, with F_{i_s} appearing in $\hat{\imath}$, by x'_{i_s}.
(b) A *counterfactual intervention* on $\mathbf{e}$ is an intervention $\hat{\imath}$ on $\mathbf{e}$, such that $L(\hat{\imath}(\mathbf{e})) = 0$. The resulting entity $\mathbf{e}' = \hat{\imath}(\mathbf{e})$ is called a *counterfactual entity* (for $\mathbf{e}$).
(c) A $\preceq$-*minimal* counterfactual intervention $\hat{\imath}$ is such that there is no counterfactual intervention $\hat{\imath}'$ on $\mathbf{e}$ with $\hat{\imath}' \prec \hat{\imath}$ (i.e. $\hat{\imath}' \preceq \hat{\imath}$, but not $\hat{\imath} \preceq \hat{\imath}'$). The resulting entity $\mathbf{e}' = \hat{\imath}(\mathbf{e})$ is called a $\preceq$-*minimal counterfactual entity*.
(d) A *counterfactual explanation* for $L(\mathbf{e})$ is a set of the form $\hat{\epsilon} = \{\langle F_{i_1}, x_{i_1}\rangle, \ldots, \langle F_{i_K}, x_{i_K}\rangle\}$, with $F_{i_j}(\mathbf{e}) = x_{i_j}$, for which there is a counterfactual intervention $\hat{\imath}(\hat{\epsilon}) = \{\langle F_{i_1}, x'_{i_1}\rangle, \ldots, \langle F_{i_K}, x'_{i_K}\rangle\}$ for $\mathbf{e}$.
(e) A counterfactual explanation $\hat{\epsilon}$ for $L(\mathbf{e})$ is $\preceq$-*minimal* if its associated counterfactual intervention $\hat{\imath}(\hat{\epsilon})$ is a $\preceq$-minimal counterfactual intervention on $\mathbf{e}$. □

Remark 3.1
Counterfactual explanations contain feature values of the original entity $\mathbf{e}$. They contain relevant information about the classification result, and interventions are used to characterize and identify them. For this reason, we will usually call an alternative entity $\mathbf{e}'$ obtained from $\mathbf{e}$ through a counterfactual intervention, a *counterfactual explanation* as well: The counterfactual explanation in the sense of Defi-

nition 3.1(d) can be read-off from $\mathbf{e}'$. □

Several minimality criteria can be expressed in terms of partial orders. We will explicitly consider two of them.

Definition 3.2
Let $\hat{\iota}_1$ and $\hat{\iota}_2$ be interventions on an entity $\mathbf{e}$. (a) $\hat{\iota}_1 \preceq^s \hat{\iota}_2$ iff $\pi_1(\hat{\iota}_1) \subseteq \pi_1(\hat{\iota}_2)$, with $\pi_1(\hat{\iota})$ the projection of $\hat{\iota}$ on the first position. (b) $\hat{\iota}_1 \preceq^c \hat{\iota}_2$ iff $|\hat{\iota}_1| \leq |\hat{\iota}_2|$. □

This definition introduces minimality under set inclusion and cardinality, resp. The former minimizes -under set inclusion- the set of features whose values are changed. The latter, the number of features that see their values changed. In the following, we will consider only these minimality criteria, mostly the second (c.f. Section 9 for a discussion).

Example 3.1
Consider three binary features $\mathcal{F} = \{F_1, F_2, F_3\}$, i.e. they take values 0 or 1. The input/output relation of a classifier $\mathcal{C}$ is shown in Table 1. Let $\mathbf{e}$ be $\mathbf{e}_1$ in the table. We want counterfactual explanations for its label 1. Any other record in the table can be seen as the result of an intervention on $\mathbf{e}_1$. However, only $\mathbf{e}_4, \mathbf{e}_7, \mathbf{e}_8$ are (results of) counterfactual interventions in that they switch the label to 0.

entity (id)	F_1	F_2	F_3	L
$\mathbf{e}_1$	0	1	1	1
$\mathbf{e}_2$	1	1	1	1
$\mathbf{e}_3$	1	1	0	1
$\mathbf{e}_4$	1	0	1	0
$\mathbf{e}_5$	1	0	0	1
$\mathbf{e}_6$	0	1	0	1
$\mathbf{e}_7$	0	0	1	0
$\mathbf{e}_8$	0	0	0	0

Table 1: Classifier $\mathcal{C}$

For example, $\mathbf{e}_4$ corresponds to the intervention $\hat{\iota}_4 = \{\langle F_1, 1\rangle, \langle F_2, 0\rangle\}$, in that $\mathbf{e}_4$ is obtained from $\mathbf{e}_1$ by changing the values of F_1, F_2 into 1 and 0, resp. For $\hat{\iota}_4$, $\pi_1(\hat{\iota}_4) = \{\langle F_1\rangle, \langle F_2\rangle\}$. From $\hat{\iota}_4$ we obtain the counterfactual explanation $\hat{\epsilon}_4 = \{\langle F_1, 0\rangle, \langle F_2, 1\rangle\}$, telling us that the values $F_1(\mathbf{e}_1) = 0$ and $F_2(\mathbf{e}_1) = 1$ are the joint cause for $\mathbf{e}_1$ to be classified as 1.

There are three counterfactual explanations: $\hat{\epsilon}_4 := \{\langle F_1, 0\rangle, \langle F_2, 1\rangle\}$, $\hat{\epsilon}_7 := \{\langle F_2, 1\rangle\}$, and $\hat{\epsilon}_8 := \{\langle F_2, 1\rangle, \langle F_3, 1\rangle\}$. Here, $\mathbf{e}_4$ and $\mathbf{e}_8$ are incomparable under $\preceq^s$, $\mathbf{e}_7 \prec^s \mathbf{e}_4$, $\mathbf{e}_7 \prec^s \mathbf{e}_8$, and $\hat{\epsilon}_7$ turns out to be $\preceq^s$- and $\preceq^c$-minimal (actually, minimum). □

Notice that what matters here is *what* is intervened, and not *how*. By taking a projection, the partial order $\preceq^s$ does not care about the values that replace the original feature values, as long as the latter are changed. Furthermore, given $\mathbf{e}$, it would be good enough to indicate the features whose values are relevant, e.g. $\hat{\epsilon}_7 = \{F_2\}$ in the previous example. However, the introduced notation emphasizes the fact that the original values are those we concentrate on when providing explanations.

Every $\preceq^c$-minimal explanation is also $\preceq^s$-minimal. However, it is easy to produce an example showing that a $\preceq^s$-minimal explanation may not be $\preceq^c$-minimal.

Notation: An *s-explanation* for $L(\mathbf{e})$ is a $\preceq^s$-minimal counterfactual explanation for $L(\mathbf{e})$. A *c-explanation* for $L(\mathbf{e})$ is a $\preceq^c$-minimal counterfactual explanation for $L(\mathbf{e})$. So as prescribed in Remark 3.1, we will usually use the terms s-explanation and c-explanation to refer to the alternative, intervened entities $\mathbf{e}'$ that are associated to s- and c-explanations in the sense of Definition 3.2. □

So far, we have characterized explanations as sets of (interventions on) features. However, one would also like to quantify the "causal strength" of a *single feature value* in a record representing an entity. Something similar has been done for a single tuple in a database as a cause for a query answer (Meliou et al. 2010), or for a single attribute value in a database tuple (Bertossi and Salimi 2017a; Bertossi 2020b). Different *scores* for feature values have been proposed in this direction, e.g. Shap in (Lundberg et al. 2020) and Resp in (Bertossi et al. 2020). Following (Chockler and Halpern 2004), we will now formulate the latter as the *responsibility* of a feature value as an *actual cause* (Halpern and Pearl 2005) for the observed classification.

Definition 3.3
Consider an entity $\mathbf{e}$ represented as a record of feature values $x_i = F_i(\mathbf{e})$, $F_i \in \mathcal{F}$.
(a) A feature value $v = F(\mathbf{e})$, with $F \in \mathcal{F}$, is a *value-explanation* for $L(\mathbf{e})$ if there is an s-explanation $\hat{\epsilon}$ for $L(\mathbf{e})$, such that $\langle F, v\rangle \in \hat{\epsilon}$.
(b) The *explanatory responsibility* of a value-explanation $v = F(\mathbf{e})$ is:

$$\mathsf{x\text{-}Resp}_{\mathbf{e},F}(v) := max\{\frac{1}{|\hat{\epsilon}|} : \hat{\epsilon} \text{ is s-explanation with } \langle F, v\rangle \in \hat{\epsilon}\}.$$

(c) If $v = F(\mathbf{e})$ is not a value-explanation, $\mathsf{x\text{-}Resp}_{\mathbf{e},F}(v) := 0$. □

Notice that (b) can be stated as $\mathsf{x\text{-}Resp}_{\mathbf{e},F}(v) := \frac{1}{|\hat{\epsilon}^\star|}$, with $\hat{\epsilon}^\star = \mathsf{argmin}\{|\hat{\epsilon}| : \hat{\epsilon}$ is s-explanation with $\langle F, v\rangle \in \hat{\epsilon}\}$.

Adopting the terminology of actual causality (Halpern and Pearl 2005), a *counterfactual value-explanation* for $\mathbf{e}$'s classification is a value-explanation v with $\mathsf{x\text{-}Resp}_{\mathbf{e},F}(v) = 1$. That is, it suffices, without company from other feature values in $\mathbf{e}$, to explain the classification. Similarly, an *actual value-explanation* for $\mathbf{e}$'s classification is a value-explanation v with $\mathsf{x\text{-}Resp}_{\mathbf{e},F}(v) > 0$. That is, v appears in an s-explanation $\hat{\epsilon}$, say as $\langle F, v\rangle$, but possibly in company of other feature values. In this case, $\hat{\epsilon} \setminus \{\langle F, v\rangle\}$ is a *contingency set* for v (c.f. (Halpern and Pearl 2005), and (Meliou et al. 2010) for the case of databases).

Example 3.2
(example 3.1 continued) $\hat{\epsilon}_7$, equivalently entity $\mathbf{e}_7$, is the only c-explanation for $\mathbf{e}_1$'s classification. Its value 1 for F_2 is a counterfactual value-explanation, and its explanatory responsibility is $\mathsf{x\text{-}Resp}_{\mathbf{e}_1,F_2}(1) := 1$. The empty set is its contingency set. Now, entity $\mathbf{e}_4$ shows that the intervened value for F_2, i.e. 1, needs $\{\langle F_1, 0\rangle\}$ as contingency set for the label to switch to 0. □

In this work we are interested mostly in c-explanations, which are our *best explanations*, and in *maximum-responsibility* feature values. Notice that maximum x-Resp scores can be obtained by concentrating only on c-explanations. Maximum-responsibility value-explanations appear in c-explanations, and only in them. C.f. Section 9 for considerations on s-explanations and features with non-maximum responsibility scores.

4 Complexity of x-Resp Computation

A natural question is about the complexity of computing the x-Resp score. Not only for the obvious reason of knowing the complexity, but also to determine if the ASP-based approach we will present is justified from the complexity point of view. We can shed some light on this matter by taking advantage of complexity results for actual causality in databases (Meliou et al. 2010; Bertossi and Salimi 2017a). It is known that there are Boolean Conjunctive Queries (BCQs), $\mathcal{Q}$, for which deciding if the responsibility of a tuple for $\mathcal{Q}$ being true in a database D is above a given threshold is *NP*-complete, in the size of D (Bertossi and Salimi 2017a).

In our case, given a fixed classifier $\mathcal{C}$, the computational problem is about deciding, for an arbitrary entity $\mathbf{e}^\star$ and rational number v, if $\mathsf{x\text{-}Resp}(\mathbf{e}^\star) > v$. The complexity is measured as a function of $|\mathbf{e}^\star|$, the size M of cartesian product of the the underlying domains, and the complexity of computing $\mathcal{C}$. (Under our ASP-based approach, M will be associated to the extensional database for the program.)

Given a relational database schema $\mathcal{S} = \{R_1, \ldots, R_k\}$, with predicates with arities $ar(R_i)$, we can see each attribute A as a feature that takes values in a finite domain $Dom(A)$. Without loss of generality, we assume the predicates R_i do not share attributes (but attributes may share a same domain). Then, we have a sequence of attributes $\langle A_1^1, \ldots, A_{ar(R_1)}^1, \ldots, A_1^k, \ldots, A_{ar(R_k)}^k \rangle$. For a database D, for each of the potential database tuples $\tau_1, \ldots \tau_M$ (in D or not), with $M = |Dom(A_1^1)| \times \cdots \times |Dom(A_{ar(R_1)}^1)| + \cdots + |Dom(A_1^k)| \times \cdots \times |Dom(A_{ar(A_k)}^k)|$, define binary features $F_j(\tau_j) := 1$ if $\tau_j \in D$, and 0, otherwise.

Now, define a binary entity $\mathbf{e}^\star = \langle F(\tau_j) \rangle_{\tau_j \in D}$. This entity, containing only 1s, represents the contents of database D, and its length coincides with the number of tuples in D, say $|D|$. Now, the population of entities is $\mathcal{E} = \{1, 0\}^{|D|}$, i.e. all the binary sequences with the same length as $\mathbf{e}^\star$. Intuitively, each entity in $\mathcal{E}$ represents a sub-instance $D^{\mathbf{e}}$ of D, by keeping only the tuples of D that are assigned the value 1. We do not need to consider the insertion of tuples in D, as will be clear soon.

Now, given the fixed BCQ $\mathcal{Q}$, for which $D \models \mathcal{Q}$, define a binary classifier $\mathcal{C}^{\mathcal{Q}}$ as follows: For $\mathbf{e} \in E$, $\mathcal{C}^{\mathcal{Q}}(\mathbf{e}) := 1$ iff $D^{\mathbf{e}} \models \mathcal{Q}$. Notice that this classifier runs in polynomial-time in the length of $\mathbf{e}$. Also, $\mathcal{C}^{\mathcal{Q}}(\mathbf{e}^\star) = 1$. The monotonicity of $\mathcal{Q}$ allows us to concentrate on sub-instances of D. Only subinstances can invalidate the query, and superinstances will always make it true. In this way, we have reduced the problem of computing the responsibility of a tuple τ_j as an actual cause for $D \models \mathcal{Q}$ to the problem of computing $\mathsf{x\text{-}Resp}_{\mathbf{e}^\star, F_j}(1)$.

Theorem 1

There is a binary polynomial-time classifier $\mathcal{C}$ over a finite set of binary entities $\mathcal{E}$ for which deciding if $\mathsf{x\text{-}Resp}_{\mathbf{e},F}(v)$ is above a certain threshold is *NP*-complete in the size of $\mathbf{e}$ plus the size of $\mathcal{E}$. □

So as in (Bertossi and Salimi 2017a), several other problems in relation to responsibility can be investigated; and it is likely that most (if not all) the results in (Bertossi and Salimi 2017a) can be used to obtain similar results for the x-Resp score.

5 Counterfactual Intervention Programs

An answer-set program has a possible-world semantics, given in terms of its *stable models*, which are the intended models of the program (Gelfond and Kahl 2014). A program consists of a set of rules with variables, possibly with negated atoms in a rule body (antecedent) and disjunctions in the head (consequent). This negation is non-monotonic, which is particulary useful for doing commonsense reasoning and specifying persistence of properties. A program has an *extensional database* consisting of ground atoms (the facts). In our case, the facts will be related to the entity under classification for whose label we want counterfactual explanations. The program specifies the possible interventions. Final, intervened versions of the original entity, that have their label switched, correspond to different stable models of the program.

Entities will be represented by means of a predicate with $n + 2$ arguments $E(\cdot; \cdots; \cdot)$. The first one holds a record (or entity) id (which may not be needed when dealing with single entities). The next n arguments hold the feature values.[2] The last argument holds an annotation constant from the set $\{\mathsf{o}, \mathsf{do}, \star, \mathsf{s}\}$. Their semantics will be specified below, by the generic program that uses them.

Initially, a record $\mathbf{e} = \langle f_1, \ldots, f_n \rangle$ has not been subject to interventions, and the corresponding entry in predicate E is of the form $E(\mathbf{e}, \bar{f}, \mathsf{o})$, where $\mathbf{e}$ is (with a bit of abuse of notation) a constant used a an entity identifier, $\bar{f}$ is an abbreviation for $f_1, \ldots, f_n$, i.e. the feature values for entity $\mathbf{e}$; and the annotation constant "o" stands for "original entity".

When the classifier gives label 1 to $\mathbf{e}$, the idea is to start changing feature values, one at a time. The intervened entity becomes then annotated with constant do in the last argument.[3] When the resulting intervened entities are classified, we may not have the classifier specified within the program. For this reason, the program uses a special predicate $\mathcal{C}(\cdots, \cdot)$, where the first arguments take the feature values of an entity under classification, and the last argument returns the binary label. This predicate can be explicitly given as a set of facts (c.f. Example 5.1), or can be specified within the program (c.f. Example 5.6), or can be invoked by the program as an external predicate (c.f. Example 5.7), much in the spirit of HEX-programs

[2] For performance-related reasons, it might be more convenient to use n 3-ary predicates to represent an entity with an identifier, but the presentation here would be more complicated.
[3] The do-operator is common to denote interventions (Pearl 2009). Here, it is just a constant.

(Eiter et al. 2019; Eiter et al. 2017). Since the original instance may have to go through several interventions until reaching one that switches the label to 0, the intermediate entities get the "transition" annotation $\star$. This is achieved by a generic program.

5.1 Counterfactual intervention programs

The generic *Counterfactual Intervention Program* (CIP) is as follows:

P1. The facts of the program are the atoms of the form $Dom_i(c)$, with $c \in Dom_i$, plus the initial entity $E(\mathsf{e}, \bar{f}, \mathsf{o})$, with $\bar{f}$ the initial vector of feature values.

P2. The transition entities are obtained as initial, original entities, or as the result of an intervention. Here, e is a variable standing for a record id.

$$E(e, \bar{x}, \star) \longleftarrow E(e, \bar{x}, \mathsf{o}) \cdot$$
$$E(e, \bar{x}, \star) \longleftarrow E(e, \bar{x}, \mathsf{do}) \cdot$$

P3. The program rule specifying that, every time the entity at hand (original or obtained after a "previous" intervention) is classified with label 1, a new value has to be picked from a domain, and replaced for the current value. The new value is chosen via the non-deterministic "choice operator", well-known in ASP (Giannotti et al. 1997). In this case, the values are chosen from the domains, subject to the condition of not being the same as the current value:

$$E(e, x_1', x_2, \ldots, x_n, \mathsf{do}) \vee \cdots \vee E(e, x_1, x_2, \ldots, x_n', \mathsf{do}) \longleftarrow E(e, \bar{x}, \star), \mathcal{C}(\bar{x}, 1), Dom_1(x_1'), \ldots, Dom_n(x_n'), x_1' \neq x_1, \ldots, x_n' \neq x_n, choice(\bar{x}, x_1'), \ldots, choice(\bar{x}, x_n') \cdot$$

In general, for each fixed $\bar{x}$, $choice(\bar{x}, y)$ chooses a unique value y subject to the other conditions in the same rule body. The use of the choice operator can be eliminated by replacing each $choice(\bar{x}, x_i')$ atom by the atom $Chosen_i(\bar{x}, x_i')$, and defining each predicate $Chosen_i$ by means of "classical" rules (Giannotti et al. 1997), as follows:[4]

$$Chosen_i(\bar{x}, y) \leftarrow E(e, \bar{x}, \star),\ \mathcal{C}(\bar{x}, 1),\ Dom_i(y),\ y \neq x_i,\ not\ DiffChoice_i(\bar{x}, y) \cdot \quad (2)$$
$$DiffChoice_i(\bar{x}, y) \leftarrow Chosen_i(\bar{x}, y'),\ y' \neq y \cdot \quad (3)$$

P4. The following rule specifies that we can "stop", hence annotation s, when we reach an entity that gets label 0:

$$E(e, \bar{x}, \mathsf{s}) \longleftarrow E(e, \bar{x}, \mathsf{do}),\ \mathcal{C}(\bar{x}, 0) \cdot$$

[4] We emphasize that we are using here the "choice operator", which is definable in ASP (as done here), and not the newer *choice rules*, which could be used here for the same purpose (and many more) and are included in the *ASP-Core-2 Standard* (Calimeri et al. 2020). We use the choice operator, because most of our programs are being run with *DLV-Complex*, which does not support choice rules.

P5. We add a em program constraint specifying that we prohibit going back to the original entity via local interventions:

$$\longleftarrow \ E(e, \bar{x}, \mathsf{do}), \ E(e, \bar{x}, \mathsf{o}).$$

P6. The counterfactual explanations can be collected by means of a predicate $Expl(\cdot, \cdot, \cdot)$ specified by means of:

$$Expl(e, i, x_i) \ \longleftarrow \ E(e, x_1, \ldots, x_n, \mathsf{o}), \ E(e, x'_1, \ldots, x'_n, \mathsf{s}), \ x_i \neq x'_i,$$

with $i = 1, \ldots, n$. They collect each value that has been changed in the original instance e, with its position in e (the second argument of the predicate). Actually, each of these is a value-explanation. □

The program will have several stable models due to the disjunctive rule and the choice operator. Each model will hold intervened versions of the original entity, and hopefully versions for which the label is switched, i.e. those with annotation s. If the classifier never switches the label, despite the fact that local interventions are not restricted, we will not find a model with a version of the initial entity annotated with s. Due to the constraint in P5., none of the models will have the original entity annotated with do, because those models would be discarded (Leone et al. 2006). The definition of the choice operator contains non-stratified negation. The semantics of ASP, which involves model minimality, makes only one of the atoms in a head disjunction true (unless forced otherwise by the program).

Example 5.1
(example 3.1 continued) Most of the CIP above is generic. Here we have the facts: $Dom_1(0)$, $Dom_1(1)$, $Dom_2(0), Dom_2(1), Dom_3(0)$, $Dom_3(1)$ and $E(\mathbf{e}_1, 0, 1, 1, \mathsf{o})$, with $\mathbf{e}_1$ a constant, the record id of the first row in Table 1. The classifier is explicitly given by Table 1. Then, predicate $\mathcal{C}$ can be specified with a set of additional facts: $\mathcal{C}(0, 1, 1, 1)$, $\mathcal{C}(1, 1, 1, 1)$, $\mathcal{C}(1, 1, 0, 1)$ $\mathcal{C}(1, 0, 1, 0)$ $\mathcal{C}(1, 0, 0, 1)$ $\mathcal{C}(0, 1, 0, 1)$ $\mathcal{C}(0, 0, 1, 0)$ $\mathcal{C}(0, 0, 0, 0)$. In them, the last entry corresponds the label assigned to the entity whose feature values are given in the first three arguments.

The stable models of the program will contain all the facts above. One of them, say $\mathcal{M}_1$, will contain (among others) the facts: $E(\mathbf{e}_1, 0, 1, 1; \mathsf{o})$ and $E(\mathbf{e}_1, 0, 1, 1; \star)$. The presence of the last atom activates rule P3., because $\mathcal{C}(0, 1, 1, 1)$ is true (for $\mathbf{e}_1$ in Table 1). New facts are produced for $\mathcal{M}_1$ (the new value due to an intervention is underlined): $E(\mathbf{e}_1, \underline{1}, 1, 1, \mathsf{do})$, $E(\mathbf{e}_1, \underline{1}, 1, 1, \star)$. Due to the last fact and the true $\mathcal{C}(1, 1, 1, 1)$, rule P3. is activated again. Choosing the value 0 for the second disjunct, atoms $E(\mathbf{e}_1, \underline{1}, \underline{0}, 1, \mathsf{do})$, $E(\mathbf{e}_1, \underline{1}, \underline{0}, 1, \star)$ are generated. For the latter, $\mathcal{C}(1, 0, 1, 0)$ is true (coming from $\mathbf{e}_4$ in Table 1), switching the label to 0. Rule P3. is no longer activated, and we can apply rule P4., obtaining $E(\mathbf{e}_1, \underline{1}, \underline{0}, 1, \mathsf{s})$.

From rule P6., we obtain explanations $Expl(\mathbf{e}_1, 1, 0), Expl(\mathbf{e}_1, 2, 1)$, showing the changed values in $\mathbf{e}_1$. All this in model $\mathcal{M}_1$. There are other models, and one of them contains $E(\mathbf{e}_1, 0, \underline{0}, 1, \mathsf{s})$, the minimally intervened version of $\mathbf{e}_1$, i.e. $\mathbf{e}_7$. □

In the next example we will show how to write and run the counterfactual inter-

vention program for Example 3.1 with the *DLV* system (Leone et al. 2006).[5] When numerical aggregations and, specially, set operations are needed, we use instead the *DLV-Complex* system (2009) (c.f. Section 9). In Example 5.7 we show a program that can be used with the newer version, *DLV2* (Alviano et al. 2017), of *DLV*, which follows the ASP-Core-2 standard (Calimeri et al. 2020).

Example 5.2
(example 5.1 continued) The answer-set program for Examples 3.1 and 5.1, written in the language for the *DLV-Complex* system is shown next. (The program portion shown right below would also run with *DLV* since it does not contain numerical aggregations.) In it, the annotation "`tr`" stands for the transition annotation "$\star$" used in Example 5.1, and `X, Xp` stand for x, x', etc. In a *DLV* program, terms starting with a lower-case letter are constants; and those stating with an upper-case letter are variables.

```
% run with C:\DLV>dlv.exe theProgram.txt > outputFile.txt
#include<ListAndSet>
#maxint = 24.
% the classifier:
cls(0,1,1,1). cls(1,1,1,1). cls(1,1,0,1). cls(1,0,1,0). cls(1,0,0,1).
cls(0,1,0,1). cls(0,0,1,0). cls(0,0,0,0).
% the domains:
dom1(0). dom1(1). dom2(0). dom2(1). dom3(0). dom3(1).
% original entity at hand:
ent(e,0,1,1,o).

% transition rules:
ent(E,X,Y,Z,tr) :- ent(E,X,Y,Z,o).
ent(E,X,Y,Z,tr) :- ent(E,X,Y,Z,do).

% admissible counterfactual interventions:
ent(E,Xp,Y,Z,do) v ent(E,X,Yp,Z,do) v ent(E,X,Y,Zp,do) :- ent(E,X,Y,Z,tr),
                                  cls(X,Y,Z,1), dom1(Xp), dom2(Yp), dom3(Zp),
                                  X != Xp, Y != Yp, Z!= Zp, chosen1(X,Y,Z,Xp),
                                  chosen2(X,Y,Z,Yp), chosen3(X,Y,Z,Zp).

% definitions of chosen operators as in equations (2) and (3):
chosen1(X,Y,Z,U) :- ent(E,X,Y,Z,tr), cls(X,Y,Z,1), dom1(U), U != X,
                    not diffchoice1(X,Y,Z,U).
diffchoice1(X,Y,Z, U) :- chosen1(X,Y,Z,Up), U != Up, dom1(U).
chosen2(X,Y,Z,U) :- ent(E,X,Y,Z,tr), cls(X,Y,Z,1), dom2(U), U != Y,
                    not diffchoice2(X,Y,Z,U).
diffchoice2(X,Y,Z, U) :- chosen2(X,Y,Z,Up), U != Up, dom2(U).
chosen3(X,Y,Z,U) :- ent(E,X,Y,Z,tr), cls(X,Y,Z,1), dom3(U), U != Z,
                    not diffchoice3(X,Y,Z,U).
diffchoice3(X,Y,Z, U) :- chosen3(X,Y,Z,Up), U != Up, dom3(U).

% stop when label has been changed:
ent(E,X,Y,Z,s) :- ent(E,X,Y,Z,do), cls(X,Y,Z,0).

% hard constraint for not returning to original entity:
:- ent(E,X,Y,Z,do), ent(E,X,Y,Z,o).

% auxiliary predicate to avoid unsafe negation in the hard constraint below:
entAux(E) :- ent(E,X,Y,Z,s).
```

[5] DLV, DLV-Complex, DLV2

```
% hard constraint for not computing models where label does not change:
:- ent(E,X,Y,Z,o), not entAux(E).

% collecting explanatory changes per argument:
expl(E,1,X) :- ent(E,X,Y,Z,o), ent(E,Xp,Yp,Zp,s), X != Xp.
expl(E,2,Y) :- ent(E,X,Y,Z,o), ent(E,Xp,Yp,Zp,s), Y != Yp.
expl(E,3,Z) :- ent(E,X,Y,Z,o), ent(E,Xp,Yp,Zp,s), Z != Zp.
```

If we run this program with DLV-Complex, we obtain the following three models; for which we do not show (most of) the original program facts, as can be requested from DLV:

```
DLV [build BEN/Jul 13 2011 gcc 4.5.2]

{ent(e,0,1,1,o), ent(e,0,1,1,tr), chosen1(0,1,1,1), chosen2(0,1,1,0),
 chosen3(0,1,1,0), ent(e,0,0,1,do), ent(e,0,0,1,tr), ent(e,0,0,1,s),
 diffchoice3(0,1,1,1), diffchoice2(0,1,1,1), diffchoice1(0,1,1,0),
 entAux(e), expl(e,2,1)}

{ent(e,0,1,1,o), ent(e,0,1,1,tr), chosen1(0,1,1,1), chosen2(0,1,1,0),
 chosen3(0,1,1,0), ent(e,0,1,0,do), ent(e,0,1,0,tr), chosen1(0,1,0,1),
 chosen2(0,1,0,0), chosen3(0,1,0,1), ent(e,0,0,0,do), ent(e,0,0,0,tr),
 ent(e,0,0,0,s), diffchoice3(0,1,0,0), diffchoice3(0,1,1,1),
 diffchoice2(0,1,0,1), diffchoice2(0,1,1,1), diffchoice1(0,1,0,0),
 diffchoice1(0,1,1,0), entAux(e), expl(e,2,1), expl(e,3,1)}

{ent(e,0,1,1,o), ent(e,0,1,1,tr), chosen1(0,1,1,1), chosen2(0,1,1,0),
 chosen3(0,1,1,0), ent(e,1,1,1,do), ent(e,1,1,1,tr), chosen1(1,1,1,0),
 chosen2(1,1,1,0), chosen3(1,1,1,0), ent(e,1,0,1,do), ent(e,1,0,1,tr),
 ent(e,1,0,1,s), diffchoice3(0,1,1,1), diffchoice3(1,1,1,1),
 diffchoice2(0,1,1,1), diffchoice2(1,1,1,1), diffchoice1(0,1,1,0),
 diffchoice1(1,1,1,1), entAux(e), expl(e,1,0), expl(e,2,1)}
```

These models correspond to the counterfactual entities $\mathbf{e}_7, \mathbf{e}_8, \mathbf{e}_4$, resp. in Example 3.1.

Notice that the program, except for the fact `ent(e,0,1,1,o)` in the 10th line, is completely generic, and can be used with any input entity that has been classified with label 1.[6] We could remove it from the program, obtaining program `theProgram2.txt`, and we could run instead

```
C:\DLV>dlv.exe ent.txt program2.txt > outputFile.txt
```

where `ent.txt` is the file containing only `ent(e,0,1,1,o).` . □

In the previous example, the classifier was given as an input/output relation, that is, as a set of facts inserted directly in the program. In other situations, we may have the classifier invoked from the program as an external predicate. In others, the classifier can be specified directly in the program, as shown in Example 2.1.

Our CIPs compute all the *counterfactual versions* (or counterfactual explanations) of the original entity $\mathbf{e}$. Each counterfactual version is represented (or characterized) by at least one of the stable models of the CIP; one that contains the

[6] If the initial label is 0 instead, no interventions would be triggered, and the only model would correspond to the initial entity.

counterfactual version of $\mathbf{e}$ annotated with "s". There may be more that one stable model associated to a counterfactual version $\mathbf{e}'$ due to the use of the choice operator. Different choices may end up leading to the same $\mathbf{e}'$.

The counterfactual explanations obtained through the CIP are not necessarily s-explanations or c-explanations (c.f. Section 3), as Example 5.2 shows. The CIPs presented so far have (only) minimal models with respect to set-inclusion of the extensions of full predicates, whereas when we compare explanations, we do it at the "attribute (or feature, or argument) level". Of course, s-explanations and c-explanations are all included among the counterfactual explanations, and are represented by stable models of the CIP. We will consider this point in Section 5.3.

5.2 Complexity of CIPs

The complexity result obtained in Section 4 makes us wonder whether using ASP-programs for specifying and computing the x-Resp score is an overkill, or, more precisely, whether they have the right and necessary expressive power and complexity to confront our problem. In fact they do. It is known that reasoning with disjunctive answer-set programs (DASP) falls in the second level of the *polynomial hierarchy* (in data complexity) (Dantsin et al. 2001), and slightly above that by the use of *weak constraints* (Leone et al. 2006) that we will use in Section 5.3. However, CIPs have the property of being *head-cycle free* (HCF), which brings down the complexity of a DASP to the first level of the polynomial hierarchy (Dantsin et al. 2001). This is in line with the result in Theorem 1.

It is easy to check that CIPs are HCF (c.f. Section 2): The ground chains in the directed graph $DG(\Pi)$ associated to a CIP Π are, due to rules P2. and P3., of the form: $E(\mathbf{e}, \bar{a}, \mathsf{do}) \rightarrow E(\mathbf{e}, \bar{a}, \star) \rightarrow E(\mathbf{e}, \bar{a}', \mathsf{do})$, with $\bar{a} \neq \bar{a}'$. They never create a cycle in the head of a ground instantiation of the disjunctive rule.

One can also see the HCF property from the fact that the CIPs become *repair-programs* (Bertossi 2011) for a database w.r.t. the integrity constraint, actually *denial constraint*, $\forall e \forall \bar{x} \neg (E(e, \bar{x}) \wedge \mathcal{C}(\bar{x}, 1))$. Denial constraints are common in databases, and their repairs and repair-programs have been investigated (Caniupan and Bertossi 2010; Bertossi 2020b). C.f. (Bertossi 2011) for additional references.

As a consequence of being HCF, a CIP can be transformed, by means of the *shift operation*, into an equivalent non-disjunctive ASP (c.f. Section 2).

Example 5.3
(example 5.1 continued) The disjunctive rule in Example 5.1 can be replaced by the three rules:

```
ent(E,Xp,Y,Z,do) :- ent(E,X,Y,Z,tr), cls(X,Y,Z,1), dom1(Xp), dom2(Yp), dom3(Zp),
                    X != Xp, Y != Yp, Z!= Zp, chosen1(X,Y,Z,Xp), chosen2(X,Y,Z,Yp),
                    chosen3(X,Y,Z,Zp), not ent(E,X,Yp,Z,do), not ent(E,X,Y,Zp,do).

ent(E,X,Yp,Z,do) :- ent(E,X,Y,Z,tr), cls(X,Y,Z,1), dom1(Xp), dom2(Yp), dom3(Zp),
                    X != Xp, Y != Yp, Z!= Zp, chosen1(X,Y,Z,Xp), chosen2(X,Y,Z,Yp),
                    chosen3(X,Y,Z,Zp), not ent(E,Xp,Y,Z,do), not ent(E,X,Y,Zp,do).

ent(E,X,Y,Zp,do) :- ent(E,X,Y,Z,tr), cls(X,Y,Z,1), dom1(Xp), dom2(Yp), dom3(Zp),
                    X != Xp, Y != Yp, Z!= Zp, chosen1(X,Y,Z,Xp), chosen2(X,Y,Z,Yp),
                    chosen3(X,Y,Z,Zp), not ent(E,Xp,Y,Z,do), not ent(E,X,Yp,Z,do).
```

The resulting program has the same answer-sets as the original program. □

5.3 C-explanations and maximum responsibility

As discussed at the end of Section 5.1, an intervened entity of the form $E(\mathbf{e}, c_1, \ldots, c_n, \mathsf{s})$, that is, representing a counterfactual explanation, may not correspond to an s- or a c-explanation. We are interested in obtaining the latter, and only them, because: (a) They are our "best explanations", and (b) They are used to define and compute the *maximum* x-Resp scores.

Moving towards computing x-Resp scores, notice that in each of the stable models $\mathcal{M}$ of the CIP, we can collect the corresponding counterfactual explanation for $\mathbf{e}$'s classification as the set $\hat{\epsilon}^{\mathcal{M}} = \{\langle F_i, c_i\rangle |\ Expl(\mathbf{e}, i; c_i) \in \mathcal{M}\}$. This can be done within a ASP system such as *DLV*, which allows set construction and aggregation, in particular, counting (Leone et al. 2006). Actually, counting comes handy to obtain the cardinality of $\hat{\epsilon}^{\mathcal{M}}$, by means of:

$$\textit{inv-resp}(\mathbf{e}, m) \longleftarrow \#count\{i : Expl(\mathbf{e}, i; c_i)\} = m. \tag{4}$$

For each model $\mathcal{M}$ of the CIP, we will obtain such a value $m(\mathcal{M})$ that shows the number of changes of feature values that lead to the associated counterfactual explanation. Notice that, in each of the models $\mathcal{M}^o$ that correspond to c-explanations, these, now minimum values $m(\mathcal{M}^o)$ will be the same, say m^o, and can be used to compute the responsibility of a feature value in $\hat{\epsilon}^{\mathcal{M}^o}$, as follows: For $Expl(\mathbf{e}, i; c_i) \in \mathcal{M}^o$,

$$\textsf{x-Resp}_{\mathbf{e},F_i}(c_i) = \frac{1}{|\hat{\epsilon}^{\mathcal{M}}|} = \frac{1}{m^o}. \tag{5}$$

Example 5.4
(example 5.2 continued) Let us add to the CIP above the rule:

```
% computing the inverse of x-Resp:
invResp(E,M) :- #count{I: expl(E,I,_)} = M, #int(M), E = e.
```

By running DLV-Complex with the new program, we obtain the models above extended with atoms representing the changes in arguments of the original entity (we omit most of the old atoms):

```
DLV [build BEN/Jul 13 2011 gcc 4.5.2]

{ent(e,0,1,1,o), ... , ent(e,0,0,1,s), expl(e,2,1), invResp(e,1)}
{ent(e,0,1,1,o), ..., ent(e,0,0,0,s), expl(e,2,1), expl(e,3,1), invResp(e,2)}
{ent(e,0,1,1,o), ..., ent(e,1,0,1,s), expl(e,1,0), expl(e,2,1), invResp(e,2)}
```

As before, we obtain three models, and each of them shows, in the last atom, the number of changes that were made to obtain the corresponding counterfactual explanation. For example, for the last model, say $\mathcal{M}_3$, corresponding to entity $\mathbf{e}_4$, we obtain $m(\mathcal{M}_3) = 2$. Similarly for the second one, corresponding to entity $\mathbf{e}_8$. The first model corresponds to the counterfactual entity $\mathbf{e}_7$ that is a c-explanation, which is shown by the minimum value "1" that predicate `inResp` takes in its second argument among all the stable models.

We can see that the first model is the one corresponding to a maximum responsibility feature value. □

In order to obtain only the models associated to c-explanations, we add *weak program constraints* to the CIP. They can be violated by a stable model of the program (unlike strong program constraints), but the *number* of violations has to be minimized. In this case, for $1 \leq i \leq n$, we add to the CIP:[7]

$$:\sim \ E(e, x_1, \ldots, x_n, \mathsf{o}),\ E(e, x'_1, \ldots, x'_n, \mathsf{s}), x_i \neq x'_i. \tag{6}$$

Only the stable models representing an intervened version of $\mathbf{e}$ with a minimum number of value discrepancies with $\mathbf{e}$ will be kept.

Example 5.5
(example 5.4 continued) The best explanations, i.e. the c-explanations, can be obtained by adding weak program constraints to the combined CIP above:

```
% weak constraints for minimizing number of changes:
:~ ent(E,X,Y,Z,o), ent(E,Xp,Yp,Zp,s), X != Xp.
:~ ent(E,X,Y,Z,o), ent(E,Xp,Yp,Zp,s), Y != Yp.
:~ ent(E,X,Y,Z,o), ent(E,Xp,Yp,Zp,s), Z != Zp.
```

If we run DLV-Complex with the extended program, we obtain a single model, corresponding to $\mathbf{e}_7$:

```
Best model: {ent(e,0,1,1,o), ent(e,0,1,1,tr), chosen1(0,1,1,1), chosen2(0,1,1,0),
             chosen3(0,1,1,0), ent(e,0,0,1,do), ent(e,0,0,1,tr), ent(e,0,0,1,s),
             diffchoice3(0,1,1,1), diffchoice2(0,1,1,1), diffchoice1(0,1,1,0),
             expl(e,2,1), entAux(e), invResp(e,1)}
Cost ([Weight:Level]): <[1:1]>
```

This model shows that counterfactual entity $\mathbf{e}_7$ has one change in the second attribute wrt. the original entity. This new entity gives a minimum inverse responsibility $m^o = 1$ to the original value in the second argument of `ent`, which leads, via (5), to its (maximum) responsibility $\mathsf{x\text{-}Resp}_{\mathbf{e}_1, F_2}(1) = \frac{1}{m^o} = 1$. □

Example 5.6
(example 2.1 continued) We present now the CIP for the classifier based on the decision-tree, in *DLV-Complex* notation. Notice that after the facts, that now do not include the classifier, we find the rule-based specification of the decision tree.

```
#include<ListAndSet>
#maxint = 24.

% facts:
   dom1(sunny). dom1(overcast). dom1(rain). dom2(high). dom2(normal).
   dom3(strong). dom3(weak).
   ent(e,sunny,normal,weak,o).   % original entity at hand

% spec of the classifier:
   cls(X,Y,Z,1) :- Y = normal, X = sunny, dom1(X), dom3(Z).
   cls(X,Y,Z,1) :- X = overcast, dom2(Y), dom3(Z).
```

[7] This notation follows the standard in (Calimeri et al. 2020; Alviano et al. 2017).

```
   cls(X,Y,Z,1) :- Z = weak, X = rain, dom2(Y).
   cls(X,Y,Z,0) :- dom1(X), dom2(Y), dom3(Z), not cls(X,Y,Z,1).

% transition rules:
   ent(E,X,Y,Z,tr) :- ent(E,X,Y,Z,o).
   ent(E,X,Y,Z,tr) :- ent(E,X,Y,Z,do).

% counterfactual rule
   ent(E,Xp,Y,Z,do) v ent(E,X,Yp,Z,do) v ent(E,X,Y,Zp,do) :-
                          ent(E,X,Y,Z,tr), cls(X,Y,Z,1), dom1(Xp), dom2(Yp),
                          dom3(Zp), X != Xp, Y != Yp, Z!= Zp,
                    chosen1(X,Y,Z,Xp), chosen2(X,Y,Z,Yp), chosen3(X,Y,Z,Zp).

% definitions of chosen operators:
   chosen1(X,Y,Z,U) :- ent(E,X,Y,Z,tr), cls(X,Y,Z,1), dom1(U), U != X,
                       not diffchoice1(X,Y,Z,U).
   diffchoice1(X,Y,Z, U) :- chosen1(X,Y,Z, Up), U != Up, dom1(U).
   chosen2(X,Y,Z,U) :- ent(E,X,Y,Z,tr), cls(X,Y,Z,1), dom2(U), U != Y,
                       not diffchoice2(X,Y,Z,U).
   diffchoice2(X,Y,Z, U) :- chosen2(X,Y,Z, Up), U != Up, dom2(U).
   chosen3(X,Y,Z,U) :- ent(E,X,Y,Z,tr), cls(X,Y,Z,1), dom3(U), U != Z,
                       not diffchoice3(X,Y,Z,U).
   diffchoice3(X,Y,Z, U) :- chosen3(X,Y,Z, Up), U != Up, dom3(U).

% Not going back to initial entity (program constraint):
   :- ent(E,X,Y,Z,do), ent(E,X,Y,Z,o).

% stop when label has been changed:
   ent(E,X,Y,Z,s) :- ent(E,X,Y,Z,do), cls(X,Y,Z,0).

% collecting changed values for each feature:
   expl(E,outlook,X)   :- ent(E,X,Y,Z,o), ent(E,Xp,Yp,Zp,s), X != Xp.
   expl(E,humidity,Y)  :- ent(E,X,Y,Z,o), ent(E,Xp,Yp,Zp,s), Y != Yp.
   expl(E,wind,Z)      :- ent(E,X,Y,Z,o), ent(E,Xp,Yp,Zp,s), Z != Zp.

   entAux(E) :- ent(E,X,Y,Z,s).        % auxiliary predicate to
                                       % avoid unsafe negation
                                       % in the constraint below
   :- ent(E,X,Y,Z,o), not entAux(E).   % discard models where
                                       % label does not change

% computing the inverse of x-Resp:
   invResp(E,M) :- #count{I: expl(E,I,_)} = M, #int(M), E = e.
```

Two counterfactual versions of **e** are obtained, as represented by the two essentially different stable models of the program, and determined by the atoms with the annotation `s` (we keep in them only the most relevant atoms, omitting initial facts and choice-related atoms):

```
{ent(e,sunny,normal,weak,o),cls(sunny,normal,strong,1),cls(sunny,normal,weak,1),
 cls(overcast,high,strong,1),cls(overcast,high,weak,1),cls(rain,high,weak,1),
 cls(overcast,normal,weak,1),cls(rain,normal,weak,1),cls(overcast,normal,strong,1),
 cls(sunny,high,strong,0),cls(sunny,high,weak,0),cls(rain,high,strong,0),
 cls(rain,normal,strong,0),ent(e,sunny,high,weak,do),ent(e,sunny,high,weak,tr),
 ent(e,sunny,high,weak,s),expl(e,humidity,normal),invResp(e,1)}

{ent(e,sunny,normal,weak,o), cls(sunny,normal,strong,1),...,
 cls(rain,normal,strong,0),ent(e,rain,normal,strong,do),ent(e,rain,normal,strong,tr),
 ent(e,rain,normal,strong,s),expl(e,outlook,sunny),expl(e,wind,weak),invResp(e,2)}
```

The first model shows the classifiers as a set of atoms, and its last line, that `ent(e,sunny,high,weak,s)` is a counterfactual version, with label 0, of the original

entity **e**, and is obtained from the latter by means of changes of values in feature Humidity, leading to an inverse score of 1. The second model shows a different counterfactual version of **e**, namely `ent(e,rain,normal,strong,s)`, now obtained by changing values for features Outlook and Wind, leading to an inverse score of 2.

Let us now add, at the end of the program the following weak constraints (labeled with `(*)`):

```
% Weak constraints to minimize number of changes:      (*)
   :~ ent(E,X,Y,Z,o), ent(E,Xp,Yp,Zp,s), X != Xp.
   :~ ent(E,X,Y,Z,o), ent(E,Xp,Yp,Zp,s), Y != Yp.
   :~ ent(E,X,Y,Z,o), ent(E,Xp,Yp,Zp,s), Z != Zp.
```

If we run the program with them, the number of changes is minimized, and we basically obtain only the first model above, corresponding to the counterfactual entity $\mathbf{e}' = ent(\mathsf{sunny}, \mathsf{high}, \mathsf{weak})$. □

As can be seen at the light of this example, more complex rule-based classifiers could be defined inside a CIP. It is also possible to invoke the classifier as an external predicate, as the following example shows.

Example 5.7
(example 5.6 continued) The program below calls the classifiers through a predicate that has an external extension, as defined by a Python program. The program has the same facts and the same rules as the the program in Example 5.6, except for a new rule that defines the classification predicate, `cls` here (and $\mathcal{C}$ in the general formulation in Section 5.1), and replaces the internal specification of the classifier:

```
cls(X,Y,Z,L) :- &classifier(X,Y,Z;L), dom1(X), dom2(Y), dom3(Z).
```

Here, the atom `&classifier(X,Y,Z;L)` corresponds to the invocation of the external classifier with parameters `X,Y,Z`, which gets an external value through variable `L`. The program was run with the version of DLV2 for Linux that supports interaction with Python, and can be downloaded from:

`https://dlv.demacs.unical.it/publications#h.cgg9mbi41jq9`

The program in Python that specifies the classifier is very simple, and it can be invoked in combination with DLV2, as follows:

`sudo ./dlv2-python-linux-x86_64 program_dlv2.txt def_class.py`.

Here, `program_dlv2.txt` is the CIP, and `def_class.py` is the very simple Python program that specifies the classifier, namely

```
def classifier(X,Y,Z):
       if (X == "sunny") and (Y == "normal"):
               return 1
       if (X == "overcast"):
               return 1
       if (X == "rain") and (Z == "weak"):
               return 1
       else:
               return 0
```

We obtain as answer-set the first one in Example 5.6. □

6 Semantic Knowledge

Counterfactual interventions in the presence of semantic conditions requires consideration. As the following example shows, not every intervention, or combination of them, may be admissible (Bertossi and Geerts 2020). In these situations declarative approaches to counterfactual interventions become particularly useful.

Example 6.1
A moving company makes automated hiring decisions based on feature values in applicants' records of the form $R = \langle appCode, ability\ to\ lift, gender, weight, height, age\rangle$. Mary, represented by $R^\star = \langle 101, 1, F, 160\ pounds, 6\ feet, 28\rangle$ applies, but is denied the job, i.e. the classifier returns: $L(R^\star) = 1$. To explain the decision, we can, hypothetically, change Mary's gender, from F into M, obtaining record $R^{\star\prime}$, for which we now observe $L(R^{\star\prime}) = 0$. Thus, her value F for *gender* can be seen as a counterfactual explanation for the initial decision.

As an alternative, we might keep the value of *gender*, and counterfactually change other feature values. However, we might be constrained or guided by an ontology containing, e.g. the denial semantic constraint $\neg(R[2] = 1 \wedge R[6] > 80)$ (2 and 6 indicating positions in the record) that prohibits someone over 80 to be qualified as fit for lifting weight. We could also have a rule, such as $(R[3] = M \wedge R[4] > 100 \wedge R[6] < 70) \rightarrow R[2] = 1$, specifying that men who weigh over 100 pounds and are younger than 70 are automatically qualified to lift weight.

In situations like this, we could add to the CIPs we had before: (a) program constraints that prohibit certain models, e.g.

$$\longleftarrow\ R(e, x, 1, y, z, u, w, \star),\ w > 80;$$

(b) additional rules, e.g.

$$R(e, x, 1, y, z, u, w, \star) \longleftarrow R(e, x, y, M, z, u, w, \star),\ z > 100, w < 70,$$

that may automatically generate additional interventions. In a similar way, one could accommodate certain preferences using weak program constraints. □

Causality and responsibility in databases in the presence of integrity constraints was introduced and investigated in (Bertossi and Salimi 2017b).

Example 6.2
(example 5.6 continued) It might be the case that in a particular region, some combinations of weather conditions are never possible, e.g. raining with a strong wind at the same time. When producing counterfactual interventions for the entity **e**, such a combination should be prohibited. This can be done by imposing a hard program constraint, that we add to the program in Example 5.6:

```
% hard constraint disallowing a particular combination    (**)
   :- ent(E,rain,X,strong,tr).
```

If we run in *DLV-Complex* the program with this constraint, but without the weak constraints labeled with `(*)` in Example 5.6, we obtain only the first model shown in

Example 5.6, corresponding to the counterfactual entity $\mathbf{e}' = ent(\mathsf{sunny}, \mathsf{high}, \mathsf{weak})$. □

As the previous example shows, we can easily impose constraints that make the counterfactual entities, or equivalently, the associated explanations, *actionable* (Ustun et al. 2019; Karimi et al. 2020b). As mentioned in Section 1, for the loan application example, we could impose a hard program constraint (i.e. add it to a CIP) of the form

$$\longleftarrow E(e, \ldots, age, \ldots, \mathsf{o}),\ E(e, \ldots, age', \ldots, *),\ age' < age, \tag{7}$$

which prevents decreasing an applicant's age.

Logic-based specifications also allow for *compilation of constraints* into rules. For example, instead of using a hard constraint, such as (7), we could directly impose the condition on a counterfactual age in the disjunctive counterfactual rule P3., of the form

$$\cdots \vee E(e, x_1, \ldots, x'_a, \ldots, \mathsf{do}) \vee \cdots \ \longleftarrow\ E(e, \bar{x}, \star), \mathcal{C}(\bar{x}, 1), \ldots, Dom_a(x'_a), \ldots, x_a < x'_a, \ldots, choice(\bar{x}, x'_a), \ldots.$$

Here, the subscript a refers to the domain and variables for the Age feature. On this basis, one could only increase the age. If this intervention leads to a successful counterfactual entity (i.e. with a positive label), we could tell the applicant that he/she has to wait to possibly succeed with the loan application.

We could also think of explicitly specifying *actionable* counterfactual entities, starting with a rule of the form

$$E(e, \bar{x}, \mathfrak{a}) \ \longleftarrow\ E(e, \bar{x}, \mathfrak{s}), \cdots,$$

where the new annotation $\mathfrak{a}$ stands for "actionable", and the rule defines an entity as actionable if it is a counterfactual (final) entity that satisfies some extra conditions.

Several possibilities offer themselves in this direction. All of them require simple, symbolic changes in the overall specification. Doing something similar with a purely procedural approach would be much more complex, and would require modifying the underlying code.

Another situation where not all interventions are admissible occurs when features take continuous values, and their domains have to be discretized. The common way of doing this, namely the combination of *bucketization and one-hot-encoding*, leads to the natural and necessary imposition of additional constraints on interventions, as we will show. Through bucketization, a feature range is discretized by splitting it into finitely many, say N, usually non-overlapping intervals. This makes the feature basically categorical (each interval becoming a categorical value). Next, through one-hot-encoding, the original feature is represented as a vector of length N of indicator functions, one for each categorical value (intervals here) (Bertossi et al. 2020). In this way, the original feature gives rise to N binary features. For example, if we have a continuous feature "External Risk Estimate" (ERE), its buckets could be: $[0, 64), [64, 71), [71, 76), [76, 81),\ [81, \infty)$. Accordingly, if for an entity $\mathbf{e}$,

$\mathsf{ERE}(\mathbf{e}) = 65$, then, after one-hot-encoding, this value is represented as the vector $[0, 1, 0, 0, 0, 0]$, because 65 falls into the second bucket.

In a case like this, it is clear that counterfactual interventions are constrained by the assumptions behind bucketization and one-hot-encoding. For example, the vector cannot be updated into, say $[0, 1, 0, 1, 0, 0]$, meaning that the feature value for the entity falls both in intervals $[64, 71)$ and $[76, 81)$. Bucketization and one-hot-encoding can make use of program constraints, such as $\longleftarrow \mathsf{ERE}(e, x, 1, y, 1, z, w, \star)$, etc. Of course, admissible interventions on predicate ERE could be easily handled with a disjunctive rule like that in P3., but without the "transition" annotation $\star$. However, the ERE record is commonly a component, or a sub-record, of a larger record containing all the feature values for an entity (Bertossi et al. 2020). Hence, the need for a more general and uniform form of specification.

Here we are considering the simple scenario in which the values are treated as unordered and categorical (binary) values. In some applications of bucketization and one-hot-encoding, one assumes and takes advantage of an underlying order inherited from the values in the buckets. Such an order could be adopted and brought into this framework by using additional rules that define that order. Developing the details is somehow outside the scope of this work.

7 Beyond Binary Features and Uncertainty

7.1 Expectation over interventions for the x-Resp *score*

The x-Resp introduced in Section 3 could be considered as a first approach to quantifying the relevance of a feature value. However, as the following example shows, we might want to go one step further.

Example 7.1
Consider a simple entity $E(\mathbf{e}; 0, a_1)$, with $0 \in Dom(F_1) = \{0, 1\}$ and $a_1 \in Dom(F_2) = \{a_1, \ldots a_k\}$. Assume that $\mathcal{C}(E(\mathbf{e}; 0, a_1)) = 1 = \mathcal{C}(E(\mathbf{e}_i; 0, a_i)) = \mathcal{C}(E(\mathbf{e}'_j; 1, a_j))$, for $1 \leq i \leq k,\ 1 \leq j \leq k-1$, but $\mathcal{C}(E(\mathbf{e}'_k; 1, a_k)) = 0$.

We can see that changing only the original first feature value does not change the label provided by the classifier. Nor does additionally changing the second feature value, except when using the last possibly value, a_k, for F_2. In this case, $\mathsf{x\text{-}Resp}_{\mathbf{e}, F_1}(0) = \frac{1}{2}$, despite the fact that almost all interventions on the second feature value do not change the label.

A similar phenomenon would appear if we had $Dom(F_1) = \{b_1, \cdots, b_k\}$, with large k, and $\mathcal{C}(E(\mathbf{e}; b_1, a_1)) = 1 = \mathcal{C}(E(\mathbf{e}_i; b_j, a_1))$, for $j = 1, \ldots, k-1$, but $\mathcal{C}(E(\mathbf{e}_k; b_k, a_1)) = 0$. In this case, the value b_1 is a counterfactual cause with explanation responsibility 1, despite the fact that most of the interventions of b_1 do not switch the label. A way to compensate for this could be taking the label average over all possible interventions. □

In order to extend the definition of the x-Resp by considering all possible interventions, we may consider the average of counterfactual labels over a given population, which would assume all entities are equally likely. In more general terms, we may

assume the underlying entity population, $\mathcal{E}$, has a probability distribution, P, which we can use to express the extended x-Resp in terms of expected values, as follows.

Consider $\mathbf{e} \in \mathcal{E}$, an entity under classification, for which $L(\mathbf{e}) = 1$, and a feature $F^\star \in \mathcal{F}$. Assume we have:

1. $\Gamma \subseteq \mathcal{F} \smallsetminus \{F^\star\}$, a set of features that may end up accompanying feature $F^\star$.
2. $\bar{w} = (w_F)_{F \in \Gamma}$, $w_F \in Dom(F)$, $w_F \neq \mathbf{e}_F$, i.e. new values for features in Γ.
3. $\mathbf{e}' := \mathbf{e}[\Gamma := \bar{w}]$, i.e. reset $\mathbf{e}$'s values for Γ as in $\bar{w}$.
4. $L(\mathbf{e}') = L(\mathbf{e}) = 1$, i.e. there is no label change with $\bar{w}$ (but maybe with an extra change for $F^\star$, in next item).
5. There is $v \in Dom(F^\star)$, with $v \neq F^\star(\mathbf{e})$ and $\mathbf{e}'' := \mathbf{e}[\Gamma := \bar{w}, F^\star := v]$.

As in Definition 3.3 and the paragraph that follows it, if $L(\mathbf{e}) \neq L(\mathbf{e}'') = 0$, $F^\star(\mathbf{e})$ is an *actual causal explanation* for $L(\mathbf{e}) = 1$, with "contingency set" $\langle \Gamma, \mathbf{e}_\Gamma \rangle$, where $\mathbf{e}_\Gamma$ is the projection of $\mathbf{e}$ on Γ.

In order to define the "local" responsibility score, make v vary randomly under conditions 1.-5.:

$$\text{x-Resp}^P(\mathbf{e}, F^\star, \Gamma, \bar{w}) := \frac{L(\mathbf{e}') - \mathbb{E}[L(\mathbf{e}'') | \mathbf{e}''_{\mathcal{F} \smallsetminus \{F^\star\}} = \mathbf{e}'_{\mathcal{F} \smallsetminus \{F^\star\}}]}{1 + |\Gamma|}. \tag{8}$$

If, as so far, label 1 is what has to be explained, then $L(\mathbf{e}') = 1$, and the numerator is a number between 0 and 1. Here, Γ is fixed. Now we can minimize its size, obtaining the (generalized) responsibility score as the maximum local value; everything relative to distribution P:

$$\begin{aligned} \text{x-Resp}^P_{\mathbf{e}, F^\star}(F^\star(\mathbf{e})) \quad := \quad & max \ \text{x-Resp}(\mathbf{e}, F^\star, \Gamma, \bar{w}) \\ & |\Gamma| \ min., \bar{w}, \ (8) > 0 \\ & \langle \Gamma, \bar{w} \rangle \models 1 \cdot\text{–}4\cdot \end{aligned} \tag{9}$$

This score was introduced, with less motivation and fewer details, and definitely not on a causal basis, in (Bertossi et al. 2020), where experiments are shown, and different probability distributions are considered.

7.2 Domain knowledge under uncertainty

Different probability distribution on the entity population $\mathcal{E}$ can be considered in the definition and computation of the generalized responsibility score (c.f. Section 7.1). A natural choice is the *uniform distribution*, P^u, that gives equal probability, $\frac{1}{|\mathcal{E}|}$, to each entity in $\mathcal{E}$. Another natural distribution is the *product distribution*, $P^\times$, that is obtained, under the assumption of independence, as the product of given marginal distributions, p_F, of the features $F \in \mathcal{F}$: $P^\times(f_1, \cdots, f_n) := \Pi_{F_i \in \mathcal{F}} p_{F_i}(f_i)$.

We can also assume we have a sample from the entity population $S \subseteq \mathcal{E}$ that is used as a proxy for the latter. In this case, the distributions above become *empirical distributions*. In the uniform case, it is given by: $\hat{P}^u(\mathbf{e}) := \frac{1}{|S|}$ if $\mathbf{e} \in S$, and 0, otherwise. In the case of the product, $\hat{P}^\times(f_1, \ldots, f_n) := \Pi_{F_i \in \mathcal{F}} \hat{p}_{F_i}(f_i)$, with $\hat{p}_{F_i}(f_i) := \frac{|\{\mathbf{e} \in S \,|\, \mathbf{e}_i = f_i\}|}{|S|}$. A discussion on the use of these distributions in the context of explanation scores can be found in (Bertossi et al. 2020).

In general, one can say that the uniform distribution may not be appropriate for capturing capturing correlations between feature values. One could argue that certain combinations of feature values may be more likely than others; or that certain correlations among them exist. This situation is aggravated by the product distribution due to the independence assumption. For these reasons an empirical distribution may be better for this purpose.

In any way, having chosen a distribution on the population, $P^\star$, to work with; in particular, to compute the expectations needed for the responsibility score in (8), one could consider modifying the probabilities in the hope of capturing correlations and logical relationships between feature values. In particular, one could introduce *constraints* that prohibit certain combinations of values, in the spirit of *denial constraints* in databases, but in this case admitting positive and negative atoms. For example, with propositional features *Old* standing for *"Is older than 20"* and *OverDr* for *"Gets an account overdraft above \$50K"*, we may want to impose the prohibition $\neg(\overline{Old} \wedge OverDr)$, standing for "*nobody under 20 gets at overdraft above \$50K*".

These constraints, which are satisfied or violated by a single entity at a time, are of the form:

$$\chi: \ \neg(\bigwedge_{F \in \mathcal{F}_1} F \wedge \bigwedge_{F' \in \mathcal{F}_2} \bar{F}'), \tag{10}$$

where $\mathcal{F}_1 \cup \mathcal{F}_2 \subseteq \mathcal{F}$, $\mathcal{F}_1 \cap \mathcal{F}_2 = \emptyset$, and $F, \bar{F}'$ mean that features F, F' take values 1 and 0, resp.

The event associated to χ is $E(\chi) = \{\mathbf{e} \in \mathcal{E} \,|\, \mathbf{e} \models \chi\}$, where $\mathbf{e} \models \chi$ has the obvious meaning of satisfaction of χ by entity $\mathbf{e}$. In order to accommodate the constraint, given the initial probability space $\langle \mathcal{E}, P^\star \rangle$, we can redefine the probability as follows. For $E \subseteq \mathcal{E}$,

$$P^\star_\chi(E) := P^\star(E|E(\chi)) = \frac{P^\star(E \cap E(\chi))}{P^\star(E(\chi))}. \tag{11}$$

If χ is consistent with the population, i.e. satisfiable in $\mathcal{E}$, the conditional distribution is well-defined. Now, the probability of χ's violation set is:

$$P^\star_\chi(\mathcal{E} \smallsetminus E(\chi)) = \frac{P^\star(\emptyset)}{P^\star(E(\chi))} = 0.$$

This definition can be extended to finite and consistent sets, Θ, of constraints, by using $P^\star_{\wedge\Theta}(E)$ in (11), with $\wedge\Theta$ the conjunction of the constraints in Θ.

Of course, one could go beyond constraints of the form (10), applying the same ideas, and consider any propositional formula that is intended to be evaluated on a single entity at a time, as opposed to considering combinations of feature values for different entities.

The resulting modified distribution that accommodates the constraints could be used in the computation of any of the scores expressed in terms of expected values or in probabilistic terms.

An alternative approach consists in restricting the (combinations of) interventions in the definitions and computation of the responsibility score, as suggested in

Section 6 (and any other score based on counterfactual interventions, as a matter of fact). It is worth performing experimental comparisons between the two approaches.

8 Related Work

In this worl we consider only *local methods* in that we are not trying to explain the overall behavior of a classifier, but a particular output on the basis of individual feature values. We also consider *model-agnostic methods* that can be applied to black-box models, i.e. without the need for an access to the internal components of the model. Of course, these approaches can be applied to open models, i.e. that make all its components transparent for analysis. Actually, in some cases, it is possible to take computational advantage of this additional source of knowledge (more on this below in this section).

There are several approaches to the explanation of outcomes from classification models. These methods can be roughly categorized as those that provide *attribution scores* and those that provide *sufficient explanations*. Those in the former class assign a number to each feature value that reflects its relevance for the outcome at hand. x-Resp falls in this category. For comparison with other approaches, we repeat that, in the case of x-Resp, one counterfactually modifies feature values to see if the outcome changes. The score is computed from those changes.

Counterfactual changes are omnipresent in attribution-score-based methods, and they can be used to consider alternative entities to the one under explanation, and a notion of distance between those alternatives and the latter (Martens and Provost 2014; Wachter et al. 2017; Russell 2019; Karimi et al. 2020a; Datta et al. 2016; Bertossi et al. 2020). Sometimes the counterfactuals are less explicit, as with the popular Shap score (Lundberg et al. 2020), that is based on the Shapley value of game theory. It can be seen as a counterfactual-based score in that all possible combinations of features values (and then, most of the time departing from the initial entity) are considered in a complex aggregation (an average or expected value).

The Shap score is designed as a model-agnostic method. However, for a large class of classifiers whose internal components can be used for the score computation, Shap becomes computable in polynomial-time, while its general computational complexity is $\#P$-hard (Arenas et al. 2021; Van den Broeck et al. 2021). As we showed in this paper, the computation of the x-Resp is NP-hard. Actually, the generalized, probabilistic extension of x-Resp (c.f. Section 7.1), for certain classifiers and probability distributions on the underlying population, x-Resp can be $\#P$-hard (Bertossi et al. 2020). An investigation of classes of classifiers for which the x-Resp score (deterministic as in this work or probabilistic) can be computed in polynomial time is still open.

The popular LIME score (Ribeiro et al. 2016) is also an attribution score. It appeals to an explainable model that locally approximates the agnostic model around the entity under classification. From the resulting approximation feature scores can be computed.

Sufficient-explanation methods try to identify those (combinations of) feature

values that alone determine the outcome at hand, in the sense that, by keeping those values and possibly changing all the others, the outcome remains the same (Ribeiro et al. 2018; Wang et al. 2021). One could say, in some sense, that the outcome is *entailed* by the values that appear in a sufficient explanation. Identifying those values is reminiscent of performing an abductive diagnosis task, as done with rule-based specifications (Eiter et al. 1997), actually (Ribeiro et al. 2018) does appeal to rule-based methods.

There are some approaches to logic-based explanations for classification. They mostly follow the *sufficient-explanation* paradigm we mentioned above. More specifically, it has been shown how to "reconstruct" certain classes of classifiers, e.g. random forests, Bayesian classifiers, and binary neural networks, as Boolean circuits (Shih et al. 2018; Shi et al. 2020; Choi et al. 2020). Once a circuit is available, one can use it in particular to obtain explanations for the outcomes of the model using methods that are, in essence, abductive (Darwiche and Hirth 2020). In this context the work presented in (Ignatiev et al. 2019; Ignatiev 2019; Izza and Marques-Silva 2021) is also relevant, in that logic-based encodings of neural networks, boosted trees, and random forests are proposed and exploited for explanation purposes. Abductive and SAT-based approaches are followed. Notice that abductive methods that generate sufficient explanations can also be the basis for score definitions and their computation. Just for the gist, if a feature value appears in the large number of sufficient explanations, then it could be assigned a large individual score.

To the best of our knowledge, none of the approaches described above, and others, are based on *logical specifications of the counterfactuals* involved in the score definition and computation. Furthermore, these are specifications that can easily adopt domain or desirable logical constraints in a seamless manner, and for combined use. Actually, the logic-based representations of complex classifiers that we just mentioned above, could be the starting point for the use of our approach. For example, a Boolean circuit can be represented as a set of rules that becomes a first component of a CIP that does the counterfactual analysis on that basis.

9 Discussion

This work is about interacting via ASP with possibly external classifiers, and reasoning about their potential inputs and outputs. The classifier is supposed to have been learned by some other means. In particular, this work is not about learning ASPs, which goes in quite a different direction (Law et al. 2019).

In this work we have treated classifiers as black-boxes that are represented by external predicates in the ASP. However, we have also considered the case of a classifier that is specified within the CIP by a set of rules, to define the classification predicate $\mathcal{C}$. This was the case of a deterministic Decision Tree. Basically, each branch from the root to a label can be represented by a rule, with the branching direction at intermediate nodes represented by values in literals in a rule body, and with the label in the rule head. Something similar can be done with Boolean Circuits used as classifiers. Actually, it is possible to represent more involved classifiers as Boolean circuits (c.f. Section 8).

Our CIPs can be easily enhanced with different extensions. For example, the feature domains can be automatically specified and computed from training or test data (Bertossi et al. 2020), or other sources. As done in (Bertossi et al. 2020) for experimental purposes and using purely procedural approaches, it is possible in our ASP setting to restrict the sizes of the contingency sets, e.g. to be of size 2 (c.f. Section 3). This can be easily done by adding a cardinality condition to the body of the disjunctive intervention rule (which is supported by *DLV-Complex* and *DLV2*). Doing this would not lead to any loss of best explanations (as long as they fall within the imposed bound), and may reduce the computational work.

It is also possible to extend CIPs to make them specify and compute the *contingency sets* (of feature values) that accompany a particular value that has been counterfactually changed (c.f. Section 3). This requires a *set-building operation*, which is provided by *DLV-Complex*. This was done in (Bertossi 2020b) to compute contingency *sets* for individual database tuples as causes for query answers.

Our specification of counterfactual explanations is in some sense *ideal*, in that the whole product space of the feature domains is considered, together with the applicability of the classifier over that space. This may be impractical or unrealistic. However, we see our proposal as a conceptual and generic specification basis that can be adapted in order to include more specific declarative practices and mechanisms.

For example, restricting the product space can be done in different manners. One can use constraints or additional conditions in rule bodies. A different approach consists in replacing the product space with the entities in a *data sample* $S \subseteq \Pi_{i=1}^{n} Dom(F_i)$. We could even assume that this sample already comes with classification labels, i.e. $S^L = \{\langle \mathbf{e}'_1, L(\mathbf{e}'_1)\rangle, \ldots, \langle \mathbf{e}'_K, L(\mathbf{e}'_K)\rangle\}$. This dataset may not be disjoint from the training dataset T (c.f. Section 3). The definition of counterfactual explanation and CIPs could be adapted to these new setting without major difficulties.

The CIPs we have introduced are reminiscent of *repair programs* that specify and compute the repairs of a database that fails to satisfy the intended integrity constraints (Caniupan and Bertossi 2010). Actually, the connection between database repairs and actual causality for query answers was established and exploited in (Bertossi and Salimi 2017a). ASPs that compute tuple-level and attribute-level causes for query answers were introduced in (Bertossi 2020b). Attribute-level causes are close to interventions of feature values, but the ASPs for the former are much simpler that those presented here, because in the database scenario, changing attribute values by nulls is good enough to invalidate the query answer (the "equivalent" in that scenario to switching the classification label here). Once a null is introduced, there is no need to take it into account anymore, and a single "step" interventions are good enough.

Our CIPs are designed to obtain general counterfactual explanations, and in particular and mainly, c-explanations. The latter are associated to minimum-size contingency sets of feature values, and, at the same time, to maximum-responsibility feature values. This is achieved via the weak constraints in (6). If we wanted to obtain the responsibility of a non-maximum-responsibility feature value, that is associated to an s-explanation that is not a c-explanation, we can remove the weak

constraints (and by so doing keeping all the models of the CIP), and pose a query under the *brave semantics* about the values of *inv-resp* in (4). An approach like this was followed in (Bertossi 2020b) for database tuples as causes for query answers.

Apropos query answering, that we haven't exploited in this work, several opportunities offer themselves. For example, we could pose a query under the brave semantics to detect if a particular feature value is ever counterfactually changed, or counterfactually changed in a best explanation. Under the skeptical semantics, we can identify feature values that change in all the counterfactual entities. Fully exploiting query answering is a matter of ongoing work.

In this work we have considered s- and c-explanations, that are associated to two specific and related minimization criteria. However, as done in abstract terms in Section 3, counterfactual explanations could be cast in terms of different optimization criteria (Karimi et al. 2020a; Russell 2019; Schleich et al. 2021). One could investigate the specification and implementation of other forms of preference, the generic $\preceq$ in Definition 3.1, by using ASPs as in (Gebser et al. 2011; Brewka et al.).

Specifying and computing the generalized, probabilistic responsibility score of Section 7.1 goes beyond the usual capabilities of ASP systems. However, it would be interesting to explore the use of *probabilistic* ASPs for these tasks (Baral et al. 2009; Lee and Yang 2017). Along a similar line, *probabilistic* ASPs could be in principle used to deal with the integration of semantic constraints with underlying probability distributions on the entity population, as described in Section 7.2. This is all matter of ongoing work.

Acknowledgements: The author has been a member of RelationalAI's Academic Network, which has been a source of inspiration for this work, and much more. Part of this work was funded by ANID - Millennium ScienceInitiative Program - Code ICN17002. Help from Jessica Zangari and Mario Alviano with information about DLV2, and from Gabriela Reyes with the DLV program runs is much appreciated.

References

Alviano, M., Calimeri, F., Dodaro, C., Fuscà, D., Leone, N., Perri, S., Ricca, F., Veltri, P. and Zangari, J. The ASP System DLV2. Proc. LPNMR 2017, Springer LNCS 10377, pp. 215-221.

Arenas, M., Pablo Barceló, P., Bertossi, L. and Monet, M. The Tractability of SHAP-scores over Deterministic and Decomposable Boolean Circuits. Proc. AAAI 2021, pp. 6670-6678.

Baral, C., Gelfond, M. and Rushton, N. Probabilistic Reasoning with Answer Sets. *Theory and Practice of Logic Programming*, 2009, 9(1):57-144.

Ben-Eliyahu, R. and Dechter, R. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics in Artificial Intelligence*, 1994, 12:53-87.

Bertossi, L. *Database Repairing and Consistent Query Answering*. Synthesis Lectures in Data Management, Morgan & Claypool, 2011.

Bertossi, L. Database Repairs and Consistent Query Answering: Origins and Further Developments. Gems of PODS paper. In Proc. PODS 2019.

Bertossi, L. and Salimi, B. From Causes for Database Queries to Repairs and Model-Based Diagnosis and Back. *Theory of Computing Systems*, 2017a, 61(1):191-232.

Bertossi, L. and Salimi, B. Causes for Query Answers from Databases: Datalog Abduction, View-Updates, and Integrity Constraints. *International Journal of Approximate Reasoning*, 2017b, 90:226-252.

Bertossi, L. An ASP-Based Approach to Counterfactual Explanations for Classification. Proc. RuleML-RR 2020, Springer LNCS 12173, 2020a, pp. 70–81.

Bertossi, L. Characterizing and Computing Causes for Query Answers in Databases from Database Repairs and Repair Programs. *Knowledge and Information Systems*, 2020b. https://doi.org/10.1007/s10115-020-01516-6.

Bertossi, L., Li, J., Schleich, M., Suciu, D. and Vagena, Z. Causality-based Explanation of Classification Outcomes. Proc. 4th International Workshop on "Data Management for End-to-End Machine Learning" (DEEM) at ACM SIGMOD, 2020. Arxiv 2003.0686.

Bertossi, L. and Geerts, F. Data Quality and Explainable AI. *ACM Journal of Data and Information Quality*, 2020, 12(2):1-9.

Brewka, G., Eiter, T. and Truszczynski, M. Answer Set Programming at a Glance. *Commun. ACM*, 2011, 54(12):92–103.

Brewka, G., Delgrande, J., Romero, J. and Schaub, T. asprin: Customizing Answer Set Preferences without a Headache. Proc. AAAI 2015, pp. 1467–1474.

Calimeri, F., Cozza, S., Ianni, G., and Leone, N. An ASP System with Functions, Lists, and Sets. Proc. LPNMR 2009, Springer LNCS 5753, pp. 483–489.

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F. and Schaub, T. ASP-Core-2 Input Language Format. *Theory and Practice of Logic Programming*, 2020, 20(2):294-309.

Caniupan, M. and Bertossi, L. The Consistency Extractor System: Answer Set Programs for Consistent Query Answering in Databases. *Data & Knowledge Engineering*, 2010, 69(6):545-572.

Chockler, H. and Halpern, J. Y. Responsibility and Blame: A Structural-Model Approach. *Journal of Artificial Intelligence Research*, 2004, 22:93-115.

Choi, A., Shih, A., Goyanka, A. and Darwiche, A. On Symbolically Encoding the Behavior of Random Forests. ArXiv 2007.01493, 2020.

Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 2001, 33(3):374-425.

Datta, A., Sen, S. and Zick, Y. Algorithmic Transparency via Quantitative Input Influence: Theory and Experiments with Learning Systems. Proc. IEEE Symposium on Security and Privacy, 2016.

Darwiche, A. and Hirth, A. On the Reasons Behind Decisions. Proc. ECAI 2020, pp. 712-720.

Eiter, T., Gottlob, G. and Leone, N. Abduction from Logic Programs: Semantics and Complexity. *Theoretical Computer Science*, 1997, 189(1-2):129-177.

Eiter, T., Germano, S., Ianni, G., Kaminski, T., Redl, C., Schüller, P. and Weinzierl, A. The DLVHEX System. *Künstliche Intelligenz*, 2019, 32(2-3):187-189.

Eiter, T., Kaminski, T., Redl, C., Schüller, P. and Weinzierl, A. Answer Set Programming with External Source Access. *Reasoning Web*, Springer LNCS 10370, 2017, pp. 204-275.

Flach, P. *Machine Learning*. Cambridge Univ. Press, 2012.

Gebser, M., Kaminski, R. and Schaub, T. Complex Optimization in Answer Set Programming. *Theory and Practice of Logic Programming*, 2011, 11(4-5):821-839.

Izza, Y. and Marques-Silva, J. On Explaining Random Forests with SAT. arXiv:2105.10278, 2021.

Gelfond, M. and Kahl, Y. *Knowledge Representation and Reasoning, and the Design of Intelligent Agents*. Cambridge Univ. Press, 2014.

Gelfond, M. and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing 9*, 365–385.

Giannotti, F., Greco, S., Sacca, D. and Zaniolo, C. Programming with Non-Determinism in Deductive Databases. *Annals of Mathematics in Artificial Intelligence*, 1997, 19(1-2):97-125.

Halpern, J. and Pearl, J. Causes and Explanations: A Structural-Model Approach: Part 1. *British J. Philosophy of Science*, 2005, 56:843-887.

Ignatiev, A., Narodytska, N. and Marques-Silva, J. Abduction-Based Explanations for Machine Learning Models. Proc. AAAI 2019, pp. 1511-1519.

Ignatiev, A. Towards Trustable Explainable AI. In Proc. IJCAI 2020, pp. 5154-5158.

Karimi, A-H., Barthe, G., Balle, B. and Valera, I. Model-Agnostic Counterfactual Explanations for Consequential Decisions. Proc. International Conference on Artificial Intelligence and Statistics (AISTATS), 2020a. ArXiv 1905.11190.

Karimi, A-H., von Kügelgen, B. J., Schölkopf, B. and Valera, I. Algorithmic Recourse under Imperfect Causal Knowledge: A Probabilistic Approach. In Proc. Annual Conference on Neural Information Processing Systems (NeurIPS), 2020b.

Law, M., Russo, A. and Broda K. Logic-Based Learning of Answer Set Programs. In *Reasoning Web. Explainable Artificial Intelligence*, Springer LNCS 11810, 2019, pp. 196-231.

Lee, J. and Yang, Z. LPMLN, Weak Constraints, and P-log. Proc. AAAI 2017, pp. 1170-1177.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Koch, C., Mateis, C., Perri, S. and Scarcello, F. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 2006, 7(3):499-562.

Lundberg , S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J., Nair, B., Katz, R., Himmelfarb, J., Bansal, N. and Lee, S-I. From Local Explanations to Global Understanding with Explainable AI for Trees. *Nature Machine Intelligence*, 2020, 2(1):56-67. ArXiv 1905.04610.

Martens, D. and Provost, F. J. Explaining Data-Driven Document Classifications. *MIS Quarterly*, 2014, 38(1):73-99.

Meliou, A., Gatterbauer, W., Moore, K. F. and Suciu, D. The Complexity of Causality and Responsibility for Query Answers and Non-Answers. Proc. VLDB 2010, pp. 34-41.

Mitchell, T. M. *Machine Learning*. McGraw-Hill, 1997.

Molnar, C. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*, 2020. `https://christophm.github.io/interpretable-ml-book`

Narodytska, N., Shrotri, A., Meel, K., Ignatiev, A. and Marques-Silva, J. Assessing Heuristic Machine Learning Explanations with Model Counting. Proc. SAT 2019.

Pearl, J. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2nd edition, 2009.

Rudin, C. Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. *Nature Machine Intelligence*, 2019, 1:206-215.

Russell, Ch. Efficient Search for Diverse Coherent Explanations. Proc. FAT 2019, pp. 20-28. arXiv:1901.04909.

Ribeiro, M. T., Singh, S. and Guestrin, C. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. Proc. KDD 2016, pp. 1135-1144.

Ribeiro, M. T., Singh, S. and Guestrin, C. Anchors: High-Precision Model-Agnostic Explanations. Proc. AAAI, 2018.

Schleich, M., Geng, Z., Zhang, Y. and Suciu, D. GeCo: Quality Counterfactual Explanations in Real Time. Proceedings of the VLDB Endowment, 14(9):1681-1693.

Shi, W., Shih, A., Darwiche, A. and Choi, A. On Tractable Representations of Binary Neural Networks. Proc. KR 2020, pp. 882-892.

Shih, A., Choi, A. and Darwiche, A. Formal Verification of Bayesian Network Classifiers. Proc. International Conference on Probabilistic Graphical Models 2018, pp. 427-438.

Ustun, B., Spangher, A. and Liu, Y. Actionable Recourse in Linear Classification. In Proc. FAT 2019, pp. 10-19.

Van den Broeck, G., Lykov, A., Schleich, M. and Suciu, D. On the Tractability of SHAP Explanations. Proc. AAAI 2021, pp. 6505-6513.

Wachter, S., Mittelstadt, B. D. and Russell, C. Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR. *Harvard Journal of Law & Technology*, 31:841, 2017. arXiv 1711.00399.

Wang, E., Khosravi, P. and Van den Broeck, G. Probabilistic Sufficient Explanations. CoRR abs/2105.10118, 2021.