Explanation Graphs for Stable Models of Labelled Logic Programs

Pedro Cabalar[1,†], Brais Muñiz[1,*,†]

[1]University of Coruña, Spain

Abstract

In this paper, we introduce the notion of an explanation graph for any model of a logic program. For each true atom in the model, the graph contains a proof that uses program rules represented by rule labels. A model may have zero, one or several explanation graphs: when it has at least one, it is called a justified model. We prove that all stable models are justified whereas, in general, the opposite does not hold, at least for disjunctive programs. With the purpose of just keeping the information that is considered to be salient, we discuss several operations on explanation graphs. These include the removal of incoming or outgoing edges of a node, but also what we define as *node forgetting*, that is, removing a node while keeping the connectivity of the rest of the graph. Then, we explain how these theoretical concepts constitute the foundation of `xclingo 2.0`, a tool for explainable Answer Set Programming (ASP) that uses, in its turn, an ASP encoding to generate explanations. The tool translates the original program into a meta-program that constitutes the core of `xclingo 2.0`. We explain this encoding and prove its soundness and completeness with respect to explanation graphs. Through a practical example, we illustrate the general use of the tool and its input language based on annotations. Finally, we show how critical these annotations can be for designing meaningful, summarised, natural language explanations.

1. Introduction

In the past few years, Artificial Intelligence (AI) systems have made great advancements, generally at the cost of increasing their scale and complexity. This has frequently meant that explaining their full behaviour is no longer embraceable by human understanding. Even for symbolic AI approaches, which are credited with the advantage of being verifiable, the number and size of possible justifications generated to explain a given result may easily exceed the capacity of human comprehension. In a recent review, Tim Miller [1] discusses the adequacy of explanation generation from the point of view of cognitive science and social psychology. The author claims that explanations generated by AI systems should be more similar to explanations provided by humans, who need not be AI specialists. One would expect, therefore, some features

ASPOCP'23: Workshop on Answer Set Programming and Other Computing Paradigms

*Corresponding author.

†These authors contributed equally.

cabalar@udc.es (P. Cabalar); brais.mcastro@udc.es (B. Muñiz)

https://www.dc.fi.udc.es/~cabalar (P. Cabalar)

0000-0001-7440-0953 (P. Cabalar); 0000-0002-9817-6666 (B. Muñiz)

of human-like explanations like being context dependent, causal, selected and contrastive, among other attributes.

A line of work initiated in [2] considered the generation of *causal graph* explanations for Answer Set Programming (ASP) [3] in the context of practical Knowledge Representation (KR). This was followed later on by a more practical approach with the implementation of a tool, `xclingo` [4], that allowed the generation of explanation trees informally related to causal graphs. The orientation introduced by `xclingo` was understanding explanations as a KR feature *per se*, rather than a debugging or formal analysis tool. When we try to produce sensible, context-dependent explanations, or we try to reduce them to relevant information, an additional effort of KR specification is needed. To facilitate such a specification, `xclingo` provides an annotation language. Even though `xclingo` was only focused on effective causes derived from the positive part of the program (`xclingo` does not deal with contrastive explanations), its application to large programs, sometimes dealing with dynamic domains, revealed some important limitations. First, `xclingo` always computed all the possible derivations for each atom, even when multiple rules could be used to derive it. In large logic programs (especially those generated automatically) the number of alternative derivations may rise exponentially. This caused that, in programs where a first answer set was obtained instantly, `xclingo` required several minutes to completely explain such answer set. Second, `xclingo` explanations contained sometimes information that should be removed since, although accurate from the point of view of the logic program derivations, it was counter-intuitive under a causal reading perspective. A third important drawback was that `xclingo` did not actually count with a formal semantics describing the explanations to be obtained, so its behaviour remained purely operational.

In this paper, we describe a formal characterisation of explanations in terms of graphs constructed with atoms and program rule labels. Under this framework, models may be *justified*, meaning that they have one or more *explanation graphs*, or *unjustified* otherwise. We prove that all stable models are justified whereas, in general, the opposite does not hold, at least for disjunctive programs. We also characterize a pair of basic operations on graphs, which we call edge pruning and node forgetting, that allow performing information filtering in the explanations. These formal definitions constitute the basis of `xclingo` 2.0, which relies on an ASP encoding to generate the explanation graphs of a given answer set of some original program. We prove the soundness and correctness of this encoding and then proceed to explain the new `xclingo` 2.0 specification language, in terms of the effects it produces on explanation graphs. We also provide an illustrating example and explain several minor variations on its annotations.

The rest of the paper is structured as follows. Section 2 contains the formal definitions for explanation graphs, their properties with respect to stable models and their manipulation for information filtering. Section 3 describes our ASP implementation and proves its soundness and completeness, including a small example as an illustration. Section 4 briefly comments on related work and, finally, Section 5 concludes the paper.

2. Explanation Graphs

We start from a finite[1] signature At, a non-empty set of propositional atoms. A *(labelled) rule* is an implication of the form:

$$\ell : p_1 \vee \cdots \vee p_m \leftarrow q_1 \wedge \cdots \wedge q_n \wedge \neg s_1 \wedge \cdots \wedge \neg s_j \wedge \neg\neg t_1 \wedge \cdots \wedge \neg\neg t_k \tag{1}$$

Given a rule r like (1), we denote its label as $Lb(r) \stackrel{\text{df}}{=} \ell$. We also call the disjunction in the consequent the *head* of r, written $Head(r)$, and denote the set of head atoms as $H(r) \stackrel{\text{df}}{=} \{p_1, \ldots, p_m\}$; the conjunction in the antecedent is called the *body* of r and denoted as $Body(r)$. We also define the positive and negative parts of the body respectively as the conjunctions $Body^+(r) \stackrel{\text{df}}{=} q_1 \wedge \cdots \wedge q_n$ and $Body^-(r) \stackrel{\text{df}}{=} \neg s_1 \wedge \cdots \wedge \neg s_j \wedge \neg\neg t_1 \wedge \cdots \wedge \neg\neg t_k$. The atoms in the positive body are represented as $B^+(r) \stackrel{\text{df}}{=} \{q_1, \ldots, q_n\}$. As usual, an empty disjunction (resp. conjunction) stands for $\bot$ (resp. $\top$). A rule r with empty head $H(r) = \emptyset$ is called a *constraint*. On the other hand, when if $H(r) = \{p\}$ is a singleton, $B^+(r) = \emptyset$ and $Body^-(r) = \top$ the rule has the form $\ell : p \leftarrow \top$ and is said to be a *fact*, simply written as $\ell : p$. We will sometimes use the abbreviation $\ell : \{p\} \leftarrow B$ to stand for $\ell : p \leftarrow B \wedge \neg\neg p$. A *(labelled) logic program* P is a set of labelled rules where no label is repeated. Note that P may still contain two rules r, r' with same body and head $Body(r) = Body(r')$ and $H(r) = H(r')$, but different labels $Lb(r) \neq Lb(r')$. A program P is *positive* if $Body^-(r) = \top$ for all rules $r \in P$. A program P is *non-disjunctive* if $|H(r)| \leq 1$ for every rule $r \in P$. Finally, P is *Horn* if it is both positive and non-disjunctive: note that this may include (positive) constraints $\bot \leftarrow B$.

A propositional interpretation I is any subset of atoms $I \subseteq At$. We say that a propositional interpretation is a *model* of a labelled program P if $I \models Body(r) \rightarrow Head(r)$ in classical logic, for every rule $r \in P$. The *reduct* of a labelled program P with respect to I, written P^I, is a simple extension of the standard reduct by [5] that collects now the *labelled* positive rules:

$$P^I \stackrel{\text{df}}{=} \{\ Lb(r) : Head(r) \leftarrow Body^+(r) \ \mid\ r \in P,\ I \models Body^-(r)\ \}$$

As usual, an interpretation I is a *stable model* (or *answer set*) of a program P if I is a minimal model of P^I. Note that, for the definition of stable models, the rule labels are irrelevant. We write $SM(P)$ to stand for the set of stable models of P.

We define the rules of a program P that *support* an atom p under interpretation I as $P[I, p] \stackrel{\text{df}}{=} \{r \in P \mid p \in H(r), I \models Body(r)\}$ that is, rules with p in the head whose body is true w.r.t. I. The next proposition proves that, given I, the rules that support p in the reduct P^I are precisely the positive parts of the rules that support p in P.

Proposition 1. *For any model $I \models P$ of a program P and any atom $p \in I$: $(P^I)[I, p] = (P[I, p])^I$.* □

Definition 1 (Explanation)**.** *Let P be a labelled program and I a classical model of P. An* explanation *G of I under P is a labelled directed graph $G = \langle I, E, \lambda\rangle$ whose vertices are the atoms in I, the edges in $E \subseteq I \times I$ connect pairs of atoms, the function $\lambda : I \rightarrow Lb(P)$ assigns a label to each atom, and G further satisfies:*

[1] We leave the study of infinite signatures for future work. This will imply explanations of infinite size, but each one should contain a finite proof for each atom.

(i) *G is acyclic*

(ii) *λ is injective*

(iii) *for every $p \in I$, the rule r such that $Lb(r) = \lambda(p)$ satisfies:*
$r \in P[I, p]$ and $B^+(r) = \{q \mid (q, p) \in E\}$. □

Condition (ii) means that there are no repeated labels in the graph, i.e., $\lambda(p) \neq \lambda(q)$ for different atoms $p, q \in I$. Condition (iii) requires that each atom p in the graph is assigned the label ℓ of some rule with p in the head, with a body satisfied by I and whose atoms in the positive body form all the incoming edges for p in the graph. Intuitively, labelling p with ℓ means that the corresponding (positive part of the) rule has been fired, "producing" p as a result. Since a label cannot be repeated in the graph, each rule can only be used to produce one atom, even though the rule head may contain more than one (when it is a disjunction). It is not difficult to see that an explanation $G = \langle I, E, \lambda \rangle$ for a model I is uniquely determined by its atom labelling λ. This is because condition (iii) about λ in Definition 1 uniquely specifies all the incoming edges for all the nodes in the graph. On the other hand, of course, not every arbitrary atom labelling corresponds to a well-formed explanation. We will sometimes abbreviate an explanation G for a model I by just using its labelling λ represented as a set of pairs of the form $\lambda(p) : p$ with $p \in I$.

Definition 2 (Justified model)**.** *We say that I is a* justified *model of a labelled program P if $I \models P$ and I has some explanation G under P. We write $JM(P)$ to stand for the set of justified models of P.*

We can observe that not all models are justified, whereas a justified model may have more than one explanation, as we illustrate next.

Example 1. *Consider the labelled logic program P*

$$\ell_1 : \ a \vee b \qquad \ell_2 : \ d \leftarrow a \wedge \neg c \qquad \ell_3 : \ d \leftarrow \neg b$$

No model $I \models P$ with $c \in I$ is justified since c does not occur in any head, and so, its support is always empty $P[I, c] = \emptyset$ and c cannot be labelled. The models of P without c are $\{b\}$, $\{a, d\}$, $\{b, d\}$ and $\{a, b, d\}$ but only the first two are justified. The explanation for $I = \{b\}$ corresponds to the labelling $\{(\ell_1 : b)\}$ (it forms a graph with a single node). Model $I = \{a, d\}$ has the two possible explanation graphs:

$$\ell_1 : a \longrightarrow \ell_2 : d \qquad\qquad \ell_1 : a \qquad \ell_3 : d$$

Model $I = \{b, d\}$ is not justified: we have no support for d given I, $P[I, d] = \emptyset$, because I satisfies neither bodies of ℓ_2 nor ℓ_3. On the other hand, model $\{a, b, d\}$ is not justified either, because $P[I, a] = P[I, b] = \{\ell_1\}$ and we cannot use the same label ℓ_1 for two different atoms a and b in a same explanation (condition (ii) in Def. 1). □

Definition 3 (Proof of an atom). *Let I be a model of a labelled program P, $G = \langle I, E, \lambda \rangle$ an explanation for I under P and let $p \in I$. The* proof *for p induced by G, written $\pi_G(p)$, is the derivation:*

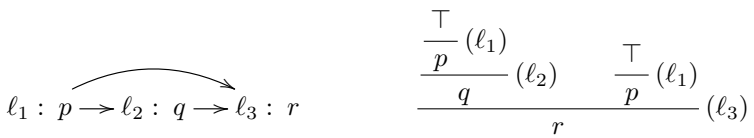
$$\pi_G(p) \stackrel{df}{=} \frac{\pi_G(q_1) \ \ldots \ \pi_G(q_n)}{p} \lambda(p),$$

where, if $r \in P$ is the rule satisfying $Lb(r) = \lambda(p)$, then $\{q_1, \ldots, q_n\} = B^+(r)$. When $n = 0$, the derivation antecedent $\pi_G(q_1) \ \ldots \ \pi_G(q_n)$ is replaced by $\top$ (corresponding to the empty conjunction). □

Example 2. *Let P be the labelled logic program:*

$$\ell_1 : \ p \qquad \ell_2 : \ q \leftarrow p \qquad \ell_3 : \ r \leftarrow p, q$$

P has a unique justified model $\{p, q, r\}$ whose explanation is shown in Figure 1 (left) whereas the induced proof for atom r is shown in Figure 1 (right). □



Explanation graph Proof for atom r

Figure 1: Some results for model $\{p, q, r\}$ of program in Example 2.

The next proposition trivially follows from the definition of explanation graphs:

Proposition 2. *If P is a Horn program, and G is an explanation for a model I of P then, for every atom, $p \in I$, $\pi_G(p)$ corresponds to a Modus Ponens derivation of p using the rules in P.*

It is worth mentioning that explanations do not generate any arbitrary Modus Ponens derivation of an atom, but only those that are globally "coherent" in the sense that, if any atom p is repeated in a proof, it is always justified repeating *the same subproof.*

In the previous examples, justified and stable models coincided: one may wonder whether this is a general property. As we see next, however, every stable model is justified but, in general, the opposite may not hold.

Theorem 1. *Stable models are justified: $SM(P) \subseteq JM(P)$.*

Proposition 3. *If P is a consistent Horn program then it has a unique justified model I that coincides with the least model of P.*

In general, the number of explanations for a single justified model can be exponential, even when the program is Horn, and so, has a unique justified and stable model corresponding to the least classical model, as we just proved. As an example:

Example 3 (A chain of firing squads). *Consider the following variation of the classical* Firing Squad Scenario *introduced by [6] for causal counterfactuals (although we do not use it for that purpose here). We have an army distributed in n squads of three soldiers each, a captain and two riflemen for each squad. We place the squads in a sequence of n consecutive hills $i = 0, \ldots, n-1$. An unfortunate prisoner is at the last hill $n-1$, and is being aimed at by the last two riflemen. At each hill i, the two riflemen a_i and b_i will fire if their captain c_i gives a signal to fire. But then, captain c_{i+1} will give a signal to fire if she hears a shot from the previous hill i in the distance. Suppose captain c_0 gives a signal to fire. Our logic program would have the form:*

$$s_0 : signal_0 \qquad \begin{array}{ll} a_i : fireA_i \leftarrow signal_i & a'_{i+1} : signal_{i+1} \leftarrow fireA_i \\ b_i : fireB_i \leftarrow signal_i & b'_{i+1} : signal_{i+1} \leftarrow fireB_i \end{array}$$

for all $i = 0, \ldots, n-1$ where we assume (for simplicity) that $signal_n$ represents the death of the prisoner. This program has one stable model (the least model) making true the $3n+1$ atoms occurring in the program. However, this last model has 2^n explanations because to derive $signal_{i+1}$ from level i, we can choose between any of the two rules a'_i or b'_i (corresponding to the two riflemen) in each explanation. □

In many disjunctive programs, justified and stable models coincide. For instance, the following example is an illustration of a program with disjunction and head cycles.

Example 4. *Let P be the program:*

$$\ell_1 : p \vee q \qquad \ell_2 : q \leftarrow p \qquad \ell_3 : p \leftarrow q$$

This program has one justified model $\{p, q\}$ that coincides with the unique stable model and has two possible explanations, $\{(\ell_1 : p), (\ell_2 : q)\}$ and $\{(\ell_1 : q), (\ell_3 : p)\}$. □

However, in the general case, not every justified model is a stable model: we provide next a simple counterexample. Consider the program P:

$$\ell_1 : a \vee b \qquad \ell_2 : a \vee c$$

whose classical models are the five interpretations: $\{a\}$, $\{a, c\}$, $\{a, b\}$, $\{b, c\}$ and $\{a, b, c\}$. The last one $\{a, b, c\}$ is not justified, since we would need three different labels and we only have two rules. Each model $\{a, c\}$, $\{a, b\}$, $\{b, c\}$ has a unique explanation corresponding to the atom labellings $\{(\ell_1 : a), (\ell_2 : c)\}$, $\{(\ell_1 : b), (\ell_2 : a)\}$ and $\{(\ell_1 : b), (\ell_2 : c)\}$, respectively. On the other hand, model $\{a\}$ has two possible explanations, corresponding to $\{(\ell_1 : a)\}$ and $\{(\ell_2 : a)\}$. Notice that, in the definition of explanation, there is no need to fire every rule with a true body in I – we are only forced to explain every true atom in I. Note also that only the justified models $\{a\}$ and $\{b, c\}$ are also stable: this is due to the minimality condition imposed by stable models on positive programs, getting rid of the other two justified models $\{a, b\}$ and $\{a, c\}$. The following theorem asserts that, for non-disjunctive programs, every justified model is also stable.

Theorem 2. *If P is a non-disjunctive program, then $SM(P) = JM(P)$.* □

In the rest of this section, we consider a pair of operations on explanation graphs that allow filtering their information depending on what a final user may consider relevant or not. These two operations are *edge pruning* and *node forgetting*. The idea behind edge pruning is as follows. Suppose we display some proof $\pi_G(p)$ as a tree, with the atom p in the root, writing the proof backwards until we reach the facts in the leaves. We may reach points in the proof where we are not interested in go on deepening, so we prefer *pruning* the tree at that node. Formally, we define this as an operation in the graph.

Definition 4 (Edge pruning). *Let $G = \langle I, E, \lambda\rangle$ be an explanation and let us define the subset of atoms $A \subseteq At$ and the subset of labels $L \subseteq Lb(P)$. Then, the* (edge) pruning *operation on G produces a new graph $prune(G, A, L) \stackrel{df}{=} \langle I,\ E \setminus E',\ \lambda\rangle$ where:*

$$E' \stackrel{df}{=} \big\{(p,q) \in E \mid p \in A\big\} \ \cup\ \{(p,q) \in E \mid \lambda(q) \in L\} \qquad \square$$

In other words, we remove the outgoing edges for all pruned atoms A and the incoming edges for all nodes q with a pruned label $\lambda(q) \in L$. As an example, Figure 2 shows in G_2 the result of pruning G_1 with $A = \{e\}$ and $L = \{\ell_4\}$.

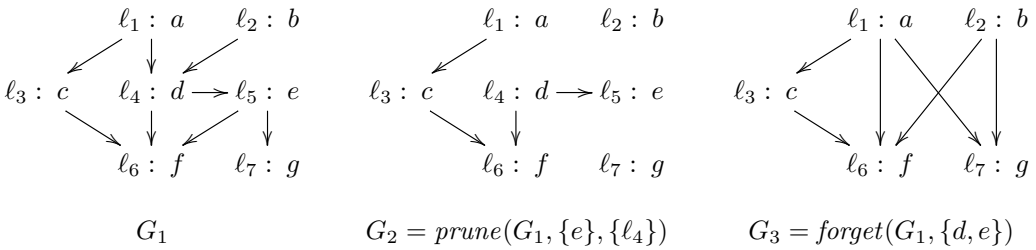


Figure 2: Examples of edge pruning and node forgetting.

The intuition for the second operation, *node forgetting*, is to obtain explanations that only consist of atoms and rules considered relevant, removing the irrelevant information.

Definition 5 (Node forgetting). *Let $G = \langle I, E, \lambda\rangle$ be an explanation and let $A \subseteq At$ be a set of atoms to be removed. Then, the* node forgetting *operation on G produces the graph $forget(G, A) \stackrel{df}{=} \langle I \setminus A,\ E',\ \lambda\rangle$ where E' contains all edges (p_0, p_n) such that there exists a path $(p_0, p_1), (p_1, p_2), \ldots, (p_{n-1}, p_n)$ in E with $n \geq 0$, $\{p_0, p_n\} \cap A = \emptyset$ and $\{p_1, \ldots, p_{n-1}\} \subseteq A$.* $\square$

Note that E' contains all original edges $(p, q) \in E$ for non-removed atoms $\{p, q\} \cap A = \emptyset$, since they correspond to the case where $n = 1$. Graph G_3 in Figure 2 is the result of forgetting nodes $A = \{d, e\}$ on G_1.

If we are going to prune some edges and forget some nodes, the order in which we perform both operations produces different results. For instance, take the graph G:

$$\ell_1 : a \ \longrightarrow\ \ell_2 : b \ \longrightarrow\ \ell_3 : c$$

and suppose we want to both prune atom $A = \{b\}$ and forget the same atom. If we start by pruning, $forget(prune(G, \{b\}, \emptyset), \{b\})$, the final result leaves two disconnected nodes $\ell_1 : a$ and $\ell_3 : c$. If we start instead by forgetting, $prune(forget(G, \{b\}), \{b\}, \emptyset)$, we get

$$\ell_1 : a \longrightarrow \ell_3 : c$$

because the result of pruning after forgetting has no effect since node b does not exist any more. For this reason, in practice, pruning will be performed in the first place (when we still have all the original nodes in the graph) and forgetting afterwards. An important observation is that, if pruning and forgetting are defined as overall operations on a set of explanations, we may get that two different explanations G_1 and G_2 end up producing the same filtered explanation G_3. For this reason, pruning and forgetting not only reduce the information in each explanation but may also significantly reduce the number of different (filtered) explanations.

3. Using ASP to compute and filter explanations: `xclingo 2.0`

One of the main features introduced in `xclingo` 2.0 is the computation of explanations based on an ASP encoding instead of the *ad hoc* algorithm used before. Given a program P, `xclingo` 2.0 builds a generic (non-ground) ASP program $x(P)$ that can be fed with the atoms in some answer set I of P to build the ground program $x(P, I)$. As we will prove, the answer sets of $x(P, I)$ are in one-to-one correspondence with the explanations of I. In this way, rather than collecting all possible explanations in a single shot, something that results too costly for explaining large programs, `xclingo` 2.0 performs regular calls to `clingo` with $x(P, I)$ to compute one, several or all explanations of I on demand. Besides, this provides a more declarative approach that can be easily extended to cover new features (such as, for instance, minimisation among explanations).

Although the real `xclingo` encoding for the program $x(P)$ is slightly more elaborated, its essential part can be just defined as follows. For each rule in P of the form (1), $x(P)$ contains the set of rules:

$$sup(\ell) \leftarrow as(q_1) \wedge \cdots \wedge as(q_n) \wedge as(p_i) \wedge \neg as(s_1) \wedge \cdots \wedge \neg as(s_j) \tag{2}$$

$$\wedge\ \neg\neg as(t_1) \wedge \cdots \wedge \neg\neg as(t_k) \tag{3}$$

$$\{f(\ell, p_i)\} \leftarrow f(q_1) \wedge \cdots \wedge f(q_n) \wedge as(p_i) \wedge sup(\ell) \tag{4}$$

$$\bot \leftarrow f(\ell, p_i) \wedge f(\ell, p_j) \tag{5}$$

for all $i, j = 1 \ldots m$ and $i \neq j$, and, additionally $x(P)$ contains the rules:

$$f(A) \leftarrow f(L, A) \wedge as(A) \tag{6}$$

$$\bot \leftarrow \mathit{not}\ f(A) \wedge as(A) \tag{7}$$

$$\bot \leftarrow f(L, A) \wedge f(L', A) \wedge L \neq L' \wedge as(A) \tag{8}$$

As we can see, $x(P)$ reifies atoms in P using three predicates: $as(A)$ which means that atom A is in the answer set I, so it is an initial assumption; $f(L, A)$ means that rule with label L has been "fired" for atom A, that is, $\lambda(A) = L$; and, finally, $f(A)$ that just means that there exists

some fired rule for A or, in other words, we were able to derive A. Predicate $sup(\ell)$ tells us that the body of the rule r with label ℓ is "supported" by I, that is, $I \models Body(r)$. Given any answer set I of P, we define the program $x(P, I) \stackrel{\text{df}}{=} x(P) \cup \{as(A) \mid A \in I\}$. It is easy to see that $x(P, I)$ becomes equivalent to the ground program containing the following rules:

$$\{f(\ell, p)\} \leftarrow f(q_1) \wedge \cdots \wedge f(q_n) \qquad \text{for each rule } r \in P \text{ like (1)}, \; I \models Body(r), p \in H(r) \cap I \tag{9}$$

$$\bot \leftarrow f(\ell, p_i) \wedge f(\ell, p_j) \qquad \text{for each rule } r \in P \text{ like (1)}, \; p_i, p_j \in H(r),\ p_i \neq p_j \tag{10}$$

$$f(a) \leftarrow f(\ell, a) \qquad \text{for each } a \in I \tag{11}$$

$$\bot \leftarrow \mathit{not}\ f(a) \qquad \text{for each } a \in I \tag{12}$$

$$\bot \leftarrow f(\ell, a) \wedge f(\ell', a) \qquad \text{for each } a \in I,\ \ell \neq \ell' \tag{13}$$

Theorem 3 (Soundness)**.** *Let I be an answer set of P. For every answer set J of program $x(P, I)$ there exists an explanation $G = \langle I, E, \lambda \rangle$ of I under P such that $\lambda(a) = \ell$ iff $f(\ell, a) \in J$.* □

Theorem 4 (Completeness)**.** *Let I be an answer set of P. For every explanation $G = \langle I, E, \lambda \rangle$ of I under P there exists some answer set J of program $x(P, I)$ where $f(\ell, a) \in J$ iff $\lambda(a) = \ell$ in G.* □

The tool `xclingo` 2.0 also performs the information filtering in an explanation, as an extension to the ASP encoding $x(P, I)$ that computes the explanations themselves. Once filtering (pruning and forgetting) is applied, we may obtain that different explanations collapse to the same filtered one: we make use of the `–project` feature in `clingo` to avoid producing repeated explanations.

To decide which atoms we want to forget or which nodes we want to prune, the `xclingo` input language allows including *annotations* in the original ASP program P. These annotations have the form of line comments beginning with `%!` followed by some keyword. As a result, `xclingo` code is fully compatible with regular `clingo`, and we may just decide to leave the annotations as usual comments (in fact, in many cases, they also help to clarify the rule meaning). Figure 3 shows an example of annotated code. The program has several facts about some weddings, divorces, and their associated years. Predicate `person(X)` just collects anybody mentioned in any wedding. Besides, predicates `wed(X,Y,D)` and `divorced(X,Y,D)` are symmetric in their arguments `X` and `Y`. Predicate `unwed(X,Y,D)` points out a wedding `wed(X,Y,D)` that became ineffective by a later divorce. Finally, predicates `married(X)` and `single(X)` indicate the current marital status of some person `X`. Note that the rule for `single(X)` is formulated as a *default*: somebody is single if we cannot prove she is currently married. This example has a unique answer set for which `xclingo` generates the unique explanation shown in Figure 4. Each tree in the output corresponds to a reversed (Modus Ponens) proof for the positive part of the program and shows the corresponding trace message at each proof node. We can see that `billy` is married because he wed `connie` in `2004`. We do not get an extended story with `billy`'s previous marriages, as they are not relevant. Similarly, `brad`, `angelina` and `jennifer` are currently single due to the application of a default: there is no current effective marriage for them. In fact, `xclingo` answers positive questions like "*why is X married?*" or

```
wed(angelina,billy,2000).    divorced(angelina,billy,2003).
wed(brad,jennifer,2000).     divorced(brad,jennifer,2005).
wed(brad,angelina,2014).     divorced(brad,angelina,2019).
wed(billy,connie,2014).

person(X) :- wed(X,Y,D).
person(Y) :- wed(X,Y,D).
%!mute {person(X)} :- person(X).

%!mute_body.
wed(X,Y,D) :- wed(Y,X,D).

divorced(X,Y,D) :- divorced(Y,X,D).
unwed(X,Y,D) :- wed(X,Y,D), divorced(X,Y,D2), D<D2.

%!trace_rule {"% is married",X}.
married(X) :- wed(X,Y,D), not unwed(X,Y,D).

%!trace_rule {"% is single, as we couldn't prove (s)he is married",X}.
single(X) :- person(X), not married(X).

%!trace {wed(X,Y,D), "% wed % in %",X,Y,D} :- wed(X,Y,D).
%!show_trace {single(X)}.
%!show_trace {married(X)}.
```

Figure 3: A program containing `xclingo` annotations.

"*why is X single?*" in terms of what has actually happened, but it does not currently consider negative questions like "*why is not* `brad` *married?*" or "*why is not* `billy` *single?*" that would require hypothetical reasoning.

The example shows how edge pruning can be specified using annotations `mute` (for atoms) and `mute_body` (for rules). For instance, the rule for the symmetric closure of `wed(X,Y,D)` is affected by `mute_body`, an annotation that must always precede the affected rule. Without this, the explanation for `connie`'s marital status would look like:

```
|__"connie is married"
|  |__"connie wed billy in 2014"
|  |  |__"billy wed connie in 2014"
```

that looks unnecessarily redundant. By muting the body of the rule, there is no real difference between `wed(X,Y,D)` and `wed(Y,X,D)` regarding its effect on the explanations. We can also observe that predicate `person(X)` is always muted. This is because, in this program, the extent of this predicate is derived from the first two arguments of `wed(X,Y,D)`. If we remove this `mute` annotation, we would get unnatural effects for single people, like for instance:

```
|__"brad is single, as we couldn't prove (s)he is married"
```

```
Answer: 1
##Explanation: 1.1
  *
  |__"angelina is single, as we couldn't prove (s)he is married"

  *
  |__"brad is single, as we couldn't prove (s)he is married"

  *
  |__"jennifer is single, as we couldn't prove (s)he is married"

  *
  |__"billy is married"
  |  |__"billy wed connie in 2014"

  *
  |__"connie is married"
  |  |__"connie wed billy in 2014"
##Total Explanations:   1
Models: 1
```

Figure 4: `xclingo` output for code in Figure 3.

```
|  |__"brad wed angelina in 2014"
```

This explanation is counter-intuitive because it uses marriage to justify that `brad` is single. Moreover, by removing the `mute` for `person` we actually get 32 explanations, since several single people have married multiple times in the past, and each marriage combination can be used in these justifications. To understand what is really happening, we may further add the following `trace` annotation:

```
%!trace {person(X),"% is a person",X} :- person(X).
```

at any point in the code to unveil the effect of predicate `person`, which was being forgotten until now. Once we do so, we obtain a more detailed justification:

```
|__"brad is single, as we couldn't prove (s)he is married"
|  |__"brad is a person"
|  |  |__"brad wed angelina in 2014"
```

that is telling us that, to decide that `brad` is single, we have used the fact that he is a person, something concluded, in its turn, from his participation in a wedding. This clarifies why predicate `person` was originally muted since the way in which we complete this predicate from others in the database should have *no causal reading* (`brad` is not a person *because* he married once).

Back to the original code in Figure 3, note how annotations `trace` and `trace_rule` are used to point out which atoms are relevant for an explanation, being all the rest automatically forgotten. For instance, we use `trace` on any atom for `wed`, so that `wed(brad,angelina,2014)` generates the trace message "`brad wed angelina in 2014`" each time this fact is included in any proof tree. The effect of `trace_rule` is similar, but is associated to the rule occurring immediately after the annotation.

4. Related work

The closest approach to the current work is clearly the already mentioned one based on *causal graphs* [2]. Although we conjecture that a formal relation can be established (we plan this for future work), the main difference is that causal graphs are "atom oriented" whereas the current approach is graph oriented. For instance, in the firing squads example, the causal-graph explanation for the derivations of atoms $signal_4$ and $signal_8$ would contain algebraic expressions with *all* the possible derivations for each one of those atoms. In the current approach, however, we would get an individual derivation in each case, but additionally, the proof we get for $signal_4$ has to be *the same one* we use for that atom inside the derivation of $signal_8$.

Justifications based on the positive part of the program were also used before in [7]. There, the authors implemented an ad-hoc approach to the problem of solving biomedical queries, rather than a general ASP explanation tool. In that paper, the authors were also able to explain unsatisfiable programs by weakening the constraints adding auxiliary head atoms. In fact, this feature has also been implemented in `xclingo` 2.0 (currently under experimentation) by adding trace messages to constraints. Another interesting feature in that approach was the selection of minimal explanations and the possibility of generating alternative explanations *different enough*, using a distance measure. These ideas can now be easily explored in `xclingo` thanks to its implementation as an ASP encoding.

Other examples of general approaches are the *formal theory of justifcations* [8], *off-line justifications* [9], LABAS [10] (based on argumentation theory [11, 12]) or s(CASP) [13]. All of them provide tree-based explanations for an atom to be (or not) in a given answer set. The formal theory of justifications was also extended to deal with justification graphs and nested justifications [14] and is actually a more general framework that allows covering other logic programming semantics. In the case of s(CASP), they provide a similar text template system to provide natural language explanations and also provide several methods for simplifying the final explanation trees. Additionally, s(CASP) proceeds in a top-down manner, building the explanation as an ordered list of literals extracted from the goal-driven satisfaction of the query. As opposed to our approach, these frameworks include the negative body of the rules as part of the explanation of why an atom is true. Although this clearly gives more detail, we claim that in practice, this orientation may become rather inconvenient (at least if we do not look for contrastive explanations). For instance, in our illustrating example, asking why `brad` is single is answered by just replying that no effective marriage was found. The previous marriages are irrelevant for that query, provided that the single status was defined as a default. This may seem an insignificant issue, but when we deal with defaults in dynamic domains, such as inertia, we may easily get too much irrelevant information. For instance, turning on a lamp at

situation 0 and asking why is it on at situation 100 may end up describing all possible ways in which the lamp was not turned off during those 100 steps, rather than just pointing out the only executed action in the initial state. We claim that negative information should be considered for contrastive queries such as "why is *not* the lamp off?" and, even in that case, a type of *minimal explanation* (as happens in diagnosis systems) would be required.

Other explanation approaches are more oriented to models as a whole, rather than to atoms in a model. For instance, [15] uses a meta-programming technique to explain why a given model *is not* an answer set of a given program. More recently, [16] considered the explanation of ASP programs that have no answer sets in terms of the concept of *abstraction* [17]. This allows spotting which parts of a given domain are actually relevant for rising the unsatisfiability of the problem.

5. Conclusions

We have provided a formal basis for version 2.0 of the ASP explanation tool `xclingo`, whose operation mode has been changed to use ASP for computing the explanations. We introduced the notion of an explanation graph for any model of a logic program and defined justified models as those that have at least one explanation. We proved that all stable models are justified, but the opposite does not hold, at least for disjunctive programs. We also defined a pair of simple operations on graphs that allow filtering their information. These definitions were then used to prove the soundness and correctness of the `xclingo` 2.0 encoding. We also explained the new features of `xclingo` language and how these are related to their formal semantics.

The main contribution of this paper is the formal definition of the explanation graphs, which can be used to implement other systems that use ASP for explanation generation, together with the proof of correctness of the `xclingo` 2.0 encoding. A more reference system description of the tool is left for a forthcoming document. It is worth mentioning that `xclingo` 2.0 is being currently used in a pair of medium/large size applications that have motivated part of the language and design changes. The first one is the tool TEXT2ALM [18], which is able to answer queries for narratives such as the ones published in the bAbI tasks [19], and is being upgraded to also explain the answer to such queries. The new system X_TEXT2ALM[2] uses `xclingo` 2.0 Python API to compute the explanations. The second application is the explanation of Decision Tree classifiers using the `crystal-tree` Python package[3] that calls `xclingo` 2.0 as a backend and has been applied to the medical domain [20].

As commented in the previous section, future work includes the explanation of unsatisfiable programs (by constraint weakening) and the minimisation or even the specification of preferences among explanations. Another interesting topic we plan to explore is different alternatives for an explanation of aggregates: right now, `xclingo` 2.0 uses all literals involved in an aggregate as a cause for its truth. Finally, we also plan to study formal comparisons to some of the approaches in the literature mentioned in the related work.

[2] https://github.com/amdorsey12/X_Text2ALMClientRelease
[3] https://github.com/bramucas/crystal-tree

Acknowledgments

Partially funded by Xunta de Galicia and the European Union, grants CITIC (ED431G 2019/01), GPC ED431B 2022/33, by the Spanish Ministry of Science and Innovation, Spain, MCIN/AEI/10.13039/501100011033 (grant PID2020-116201GB-I00) and by Project LIANDA by Fundación BBVA scientific research projects, Spain

References

[1] T. Miller, Explanation in artificial intelligence: Insights from the social sciences, Artificial Intelligence 267 (2019) 1–38.
[2] P. Cabalar, J. Fandinno, M. Fink, Causal graph justifications of logic programs, Theory and Practice of Logic Programming 14 (2014) 603–618.
[3] G. Brewka, T. Eiter, M. Truszczyński, Answer set programming at a glance, Communications of the ACM 54 (2011) 92–103.
[4] P. Cabalar, J. Fandinno, B. Muñiz, A system for explainable answer set programming, in: F. R. et al (Ed.), Proc. 36th Intl. Conf. on Logic Programming (Technical Communications), UNICAL, Rende, Italy, 2020, volume 325 of *EPTCS*, 2020, p. 13.
[5] M. Gelfond, V. Lifschitz, The stable models semantics for logic programming, in: Proc. of the 5th Intl. Conf. on Logic Programming, 1988, pp. 1070–1080.
[6] J. Pearl, Reasoning with cause and effect, in: T. Dean (Ed.), Proc. of the 16th Intl. Joint Conf. on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, Morgan Kaufmann, 1999, p. 13.
[7] E. Erdem, U. Oztok, Generating explanations for complex biomedical queries, Theory and Practice of Logic Programming 15 (2013).
[8] M. Denecker, G. Brewka, H. Strass, A formal theory of justifications, in: F. Calimeri, G. Ianni, M. Truszczynski (Eds.), Logic Programming and Nonmonotonic Reasoning - 13th Intl. Conf., LPNMR 2015, Lexington, KY, USA, 2015. Proceedings, volume 9345 of *Lecture Notes in Computer Science*, Springer, 2015, p. 15.
[9] E. Pontelli, T. C. Son, Justifications for logic programs under answer set semantics, in: S. Etalle, M. Truszczyński (Eds.), Logic Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 196–210.
[10] C. Schulz, F. Toni, Justifying answer sets using argumentation, Theory and Practice of Logic Programming 16 (2016) 59–110.
[11] A. Bondarenko, P. Dung, R. Kowalski, F. Toni, An abstract, argumentation-theoretic approach to default reasoning, Artificial Intelligence 93 (1997) 63–101.
[12] P. Dung, R. Kowalski, F. Toni, Assumption-based argumentation, Argumentation in Artificial Intelligence (2009) 199–218.
[13] J. Arias, M. Carro, Z. Chen, G. Gupta, Justifications for goal-directed constraint answer set programming, in: Intl. Conf. on Logic Programming, ICLP, 2020, p. 14.
[14] S. Marynissen, Advances in Justification Theory, Ph.D. thesis, Department of Computer Science, KU Leuven, 2022. Denecker, Marc and Bart Bogaerts (supervisors).
[15] M. Gebser, J. Pührer, T. Schaub, H. Tompits, A meta-programming technique for debugging

answer-set programs, in: D. Fox, C. P. Gomes (Eds.), Proc. of the 23rd AAAI Conf. on Artificial Intelligence, Chicago, IL, USA, AAAI Press, 2008, p. 6.

[16] T. Eiter, Z. Saribatur, P. Schüller, Abstraction for zooming-in to unsolvability reasons of grid-cell problems, in: Intl. Joint Conf. on Artificial Intelligence IJCAI 2019, Workshop on Explainable Artificial Intelligence, 2019, p. 15.

[17] Z. G. Saribatur, T. Eiter, P. Schüller, Abstraction for non-ground answer set programs, Artificial Intelligence 300 (2021) 103563.

[18] C. Olson, Y. Lierler, Information extraction tool Text2ALM: From narratives to action language system descriptions, in: Intl. Conf. on Logic Programming, ICLP, 2019, p. 20.

[19] J. Weston, A. Bordes, S. Chopra, T. Mikolov, Towards AI-complete question answering: A set of prerequisite toy tasks, in: Y. Bengio, Y. LeCun (Eds.), 4th Intl. Conf. on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conf. Track Proceedings, 2016, p. 21.

[20] P. Cabalar, B. Muñiz, G. Pérez, F. Suárez, Explainable machine learning for liver transplantation, in: 2nd Intl. Workshop on eXplainable Artificial Intelligence in Healthcare, XAI-Healthcare 2022, Rochester, Minnesota, USA, 2022, p. 11.

Appendix. Proofs

Proof of Proposition 1.

We prove first $\supseteq$: suppose $r \in P[I,p]$ and let us call $r' = Lb(r) : Head(r) \leftarrow Body^+(r)$. Then, by definition, $I \models Body(r)$ and, in particular, $I \models Body^-(r)$, so we conclude $r' \in P^I$. To see that $r' \in P^I[I,p]$, note that $I \models Body(r)$ implies $I \models Body^+(r) = Body(r')$.

For the $\subseteq$ direction, take any $r' \in P^I[I,p]$. By definition of reduct, we know that r' is a positive rule and that there exists some $r \in P$ where $Lb(r) = Lb(r')$, $H(r) = H(r')$, $B^+(r) = B^+(r')$ and $I \models Body^-(r)$. Consider any rule r satisfying that condition (we could have more than one): we will prove that $r \in P[I,p]$. Since $r' \in P^I[I,p]$, we get $I \models Body(r')$ but this is equivalent to $I \models Body^+(r)$. However, as we had $I \models Body^-(r)$, we conclude $I \models Body(r)$ and so r is supported in P given I. □

To prove Theorem 1 We start proving a correspondence between explanations for any model I of P and explanations under P^I.

Proposition 4. *Let I be a model of program P. Then G is an explanation for I under P iff G is an explanation for I under P^I.*

Proof. By Proposition 1, for any atom $p \in I$, the labels in $P[I,p]$ and $P^I[I,p]$ coincide, so there is no difference in the ways in which we can label p in explanations for P and for P^I. On the other hand, the rules in $P^I[I,p]$ are the positive parts of the rules in $P[I,p]$, so the graphs we can form are also the same. □

Corollary 1. *$I \in JM(P)$ iff $I \in JM(P^I)$.*

Proof of Theorem 1.

Let I be a stable model of P. To prove that there is an explanation G for I under P, we can use Proposition 1 and just prove that there is some explanation G for I under P^I. We will build the explanation with a non-deterministic algorithm where, in each step i, we denote the graph G_i as $G_i = \langle I_i, E_i, \lambda_i \rangle$ and represent the labelling λ_i as a set of pairs of the form $(\ell : p)$ meaning $\ell = \lambda(p)$. The algorithm proceeds as follows:

1: $I_0 \leftarrow \emptyset$; $E_0 \leftarrow \emptyset$; $\lambda_0 \leftarrow \emptyset$
2: $G_0 = \langle I_0, E_0, \lambda_0 \rangle$
3: $i \leftarrow 0$
4: **while** $I_i \not\models P^I$ **do**
5: Pick a rule $r \in P^I$ s.t. $I_i \models Body(r) \wedge \neg Head(r)$
6: Pick an atom $p \in I \cap H(r)$
7: $I_{i+1} \leftarrow I_i \cup \{p\}$
8: $\lambda_{i+1} \leftarrow \lambda_i \cup \{(\ell : p)\}$
9: $E_{i+1} \leftarrow E_i \cup \{(q,p) \mid q \in B^+(r)\}$
10: $G_i \leftarrow \langle I_i, E_i, \lambda_i \rangle$

11: $i \leftarrow i + 1$
12: **end while**

The existence of a rule $r \in P^I$ in line 5 is guaranteed because the **while** condition asserts $I_i \not\models P^I$ and so there must be some rule whose positive body is satisfied by I_i but its head is not satisfied. We prove next that the existence of an atom $p \in I \cap Head(r)$ (line 5) is also guaranteed. First, note that the **while** loop maintains the invariant $I_i \subseteq I$, since $I_0 = \emptyset$ and I_i only grows with atoms p (line 7) that belong to I (line 6). Therefore, $I_i \models Body(r)$ implies $I \models Body(r)$, but since $I \models P^I$, we also conclude $I \models r$ and thus $I \models Head(r)$ that is $I \cap H(r) \neq \emptyset$, so we can always pick some atom p in that intersection. Now, note that the algorithm stops because, in each iteration, I_i grows with exactly one atom from I that was not included before, since $I_i \models \neg Head(r)$, and so, this process will stop provided that I is finite. The **while** stops satisfying $I_i \models P^I$ for some value $i = n$. Moreover, $I_n = I$, because otherwise, as $I_i \subseteq I$ is an invariant, we would conclude $I_n \subset I$ and so I would not be a minimal model of P^I, which contradicts that I is a stable model of P. We remain to prove that the final $G_n = \langle I_n, E_n, \lambda_n \rangle$ is a correct explanation graph for I under P^I. As we said, the atoms in I are the graph nodes $I_n = I$. Second, we can easily see that G_n is acyclic because each iteration adds a new node p and links this node to previous atoms from $B^+(r) \subseteq I_i$ (remember $I_i \models Body(r)$) so no loop can be formed. Third, no rule label can be repeated, because we go always picking a rule r that is new, since it was not satisfied in I_i but becomes satisfied in I_{i+1} (the rule head $Head(r)$ becomes true). Last, for every $p \in I$, it is not hard to see that the (positive) rule $r \in P^I$ such that $Lb(r) = \lambda_n(p)$ satisfies $p \in H(r)$ and $B^+(r) = \{q \mid (q, p) \in E\}$ by the way in which we picked r and inserted p in I_i, whereas $I \models Body(r)$ because $I_i \models Body(r)$, r is a positive rule and $I_i \subseteq I$. □

Proof of Proposition 3.

Since P is Horn and consistent (all constraints are satisfied) its unique stable model is the least model I. By Theorem 1, I is also justified by some explanation G. We remain to prove that I is the unique justified model. Suppose there is another model $J \supset I$ (remember I is the least model) justified by an explanation G and take some atom $p \in J \setminus I$. Then, by Proposition 2, the proof for p induced by G, $\pi_G(p)$, is a Modus Ponens derivation of p using the rules in P. Since Modus Ponens is sound and the derivation starts from facts in the program, this means that p must be satisfied by any model of P, so $p \in I$ and we reach a contradiction. □

Proof of Theorem 2.

Given Theorem 1, we must only prove that, for non-disjunctive programs, every justified model is also stable. Let I be a justified model of P. By Proposition 4, we also know that I is a justified model of P^I. P^I is a positive program and is non-disjunctive (since P was non-disjunctive) and so, P is a Horn program. By Proposition 3, we know I is also the *least model* of P^I, which makes it a stable model of P. □

Proof of Theorem 3.

We have to prove that J induces a valid explanation graph G. Let us denote $At(J) \stackrel{\text{df}}{=} \{a \in At \mid f(a) \in J\}$. Since (11) is the only rule for $f(a)$, we can apply completion to conclude

that $f(a) \in J$ iff $f(\ell, a) \in J$ for some label ℓ. So, the set $At(J)$ contains the set of atoms for which J assigns some label: we will prove that this set coincides with I. We may observe that $I \subseteq At(J)$ because for any $a \in I$ we have the constraint (12) forcing $f(a) \in J$. On the other hand, $At(J) \subseteq I$ because the only rules with $f(a)$ in the head are (11) and these are only defined for atoms $a \in I$. To sum up, in any answer set J of $x(P, I)$, we derive exactly the original atoms in I, $At(J) = I$ and so, the graph induced by J has exactly one node per atom in I.

Constraint (13) guarantees that atoms $f(\ell, a)$ have a functional nature, that is, we never get two different labels for a same atom a. This allows defining the labelling function $\lambda(a) = \ell$ iff $f(\ell, a) \in J$. We remain to prove that conditions (i)-(iii) in Definition 1 hold. Condition (ii) requires that λ is injective, something guaranteed by (10). Condition (iii) requires that, informally speaking, the labelling for each atom a corresponds to an activated, supported rule for a. That is, if $\lambda(a) = \ell$, or equivalently $f(\ell, a)$, we should be able to build am edge (q, a) for each atom in the positive body of ℓ so that atoms q are among the graph nodes. This is guaranteed by that fact that rule (9) is the only one with predicate $f(\ell, a)$ in the head. So, if that ground atom is in J, it is because $f(q_i)$ are also in J i.e. $q_i \in I$, for all atoms in the positive body of rule labelled with ℓ. Note also that (9) is such that $I \models Body(r)$, so the rule supports atom p under I, that is, $r \in P[I, p]$. Let us call E to the set of edges formed in this way. Condition (i) requires that the set E of edges forms an acyclic graph. To prove this last condition, consider the reduct program $x(P, I)^J$. The only difference of this program with respect to $x(P, I)$ is that rules (9) have now the form:

$$f(\ell, p) \leftarrow f(q_1) \wedge \cdots \wedge f(q_n) \tag{14}$$

for each rule $r \in P$ like (1), $I \models Body(r), p \in H(r) \cap I$ as before, but additionally $f(\ell, p) \in J$ so the rule is kept in the reduct. Yet, the last condition is irrelevant since $f(\ell, p) \in J$ implies $f(p) \in J$ so $p \in At(J) = I$. Thus, we have exactly one rule (14) in $x(P, I)^J$ per each choice (9) in $x(P, I)$. Now, since J is an answer set of $x(P, I)$, by monotonicity of constraints, it (10), (12) and (13) and is an answer set of the rest of the program P' formed by rules (14) and (10). This means that J is a minimal model of P'. Suppose we have a cycle in E, formed by the (labelled) nodes and edges $(\ell_1 : p_1) \longrightarrow \cdots \longrightarrow (\ell_n : p_n) \longrightarrow (\ell_1 : p_1)$.Take the interpretation $J' = J \setminus \{f(\ell_1, p_1), \ldots, f(\ell_n, p_n), f(p_1), \ldots, f(p_n)\}$. Since J is a minimal for P' there must be some rule (14) or (10) not satisfied by J'. Suppose J' does not satisfy some rule (10) so that $f(a) \notin J'$ but $f(\ell, a) \in J' \subseteq J$. This means we had $f(a) \in J$ since the rule was satisfied by J so a is one of the removed atoms p_i belonging to the cycle. But then $f(\ell, a)$ should have been removed $f(\ell, a) \notin J'$ and we reach a contradiction. Suppose instead that J' does not satisfy some rule (14), that is, $f(\ell, p) \notin J'$ and $\{f(q_1), \ldots, f(g_n)\} \subseteq J' \subseteq J$. Again, since the body holds in J, we get $f(\ell, p) \in J$ and so, $f(\ell, p)$ is one of the atoms in the cycle we removed from J'. Yet, since $(\ell : p)$ is in the cycle, there is some incoming edge from some atom in the cycle and, due to the way in which atom labelling is done, this means that this edge must come from some atom q_i with $1 \leq i \leq n$ in the positive body of the rule whose label is ℓ. But, since this atom is in the cycle, this also means that $f(q_i) \notin J'$ and we reach a contradiction. □

Proof of Theorem 4.

Take I an answer set of P and $G = \langle I, E, \lambda \rangle$ some explanation for I under P and let us define

the interpretation:

$$J := \{f(a) \mid a \in I\} \cup \{f(\ell, a) \mid \lambda(a) = \ell\}$$

We will prove that J is an answer set of $x(P, I)$ or, in other words, that J is a minimal model of $x(P, I)^J$. First, we will note that J satisfies $x(P, I)^J$ rule by rule. For the constraints, J obviously satisfy (6) because it contains an atom $f(a)$ for each $a \in I$. We can also see that J satisfies (10) because graph G does not contain repeated labels, so we cannot have two different atoms with the same label. The third constraint (13) is also satisfied by J because atoms $f(\ell, a), f(\ell', a)$ are obtained from $\lambda(a)$ that is a function that cannot assign two different labels to a same atom a. Satisfaction of (10) is guaranteed since the head of this rule $f(a)$ is always some atom $a \in I$ and therefore $f(a) \in J$. For the remaining rule, (9), we have two cases. If $f(\ell, p) \notin J$ then the rule is not included in the reduct and so there is no need to be satisfied. Otherwise, if $f(\ell, p) \in J$ then the rule in the reduct corresponds to (14) and is trivially satisfied by J because its only head atom holds in that interpretation. Finally, to prove that J is a minimal model of $x(P, I)^J$, take the derivation tree $\pi_G(a)$ for each atom $a \in I$. Now, construct a new tree π where we replace each atom p in $\pi_G(a)$ by an additional derivation from $f(\ell, p)$ to $f(p)$ through rule (11). It is easy to see that π constitutes a Modus Ponens proof for $f(a)$ under the Horn program $x(P, I)^J$ and the same reasoning can be applied to atom $f(\ell, a) \in J$ that is derived in the tree π for $f(a)$. Therefore, all atoms in J must be included in any model of $x(P, I)^J$. □