TPLP **24** (6): 1109–1122, 2024.

doi:10.1017/S1471068424000048 First published online 29 February 2024

Model Explanation via Support Graphs

PEDRO CABALAR and BRAIS MUÑIZ
University of A Coruña, Spain
(*e-mails:* cabalar@udc.es, brais.mcastro@udc.es)

submitted 9 August 2023; revised 28 December 2023; accepted 8 January 2024

Abstract

In this note, we introduce the notion of support graph to define explanations for any model of a logic program. An explanation is an acyclic support graph that, for each true atom in the model, induces a proof in terms of program rules represented by labels. A classical model may have zero, one or several explanations: when it has at least one, it is called a justified model. We prove that all stable models are justified, whereas, for disjunctive programs, some justified models may not be stable. We also provide a meta-programming encoding in Answer Set Programming that generates the explanations for a given stable model of some program. We prove that the encoding is sound and complete, that is, there is a one-to-one correspondence between each answer set of the encoding and each explanation for the original stable model.

KEYWORDS: answer set programming, explanations, supported models, justified models

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. Although symbolic AI approaches have 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. Consider, for instance, the case of Answer Set Programming (ASP) (Brewka *et al.* 2011), a successful logic programming paradigm for practical Knowledge Representation and problem-solving. Even for a positive program, whose answer set is unique, the number of proofs for an atom we can form using *modus ponens* can be exponential. It makes sense, then, to generate explanations through the typical ASP problem-solving orientation. Namely, we may consider each explanation *individually* as one solution to the "explainability problem" (i.e., explaining a model) and let the user decide to generate one, several or all of them, or perhaps to impose additional preference conditions as done with optimisation problems in ASP.

In this technical note, 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 *explanations*, 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 provide an ASP encoding to generate

the explanations of a given answer set of some original program, proving the soundness and completeness of this encoding.

The rest of this note is structured as follows. Section 2 contains the formal definitions for explanations and their properties with respect to stable models. Section 3 describes the ASP encoding and proves its soundness and completeness. Section 4 briefly comments on related work and, finally, Section 5 concludes the paper.

2 Explanations as support 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 $p_1 \vee \cdots \vee p_m$ 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 $H(r) = \{p\}$ 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 $\ell : p$. The use of double negation in the body allows representing elementary choice rules. For instance, 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 *(classical) 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 Gelfond and Lifschitz (1988) 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 $SUP(P, I, p) \stackrel{\text{df}}{=} \{r \in P \mid p \in H(r), I \models Body(r)\}$ that is, rules with p in the head whose

[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.

https://doi.org/10.1017/S1471068424000048 Published online by Cambridge University Press

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 $p \in I$: $SUP(P^I, I, p) = SUP(P, I, p)^I$.

Proof
We prove first $\supseteq$: suppose $r \in SUP(P, I, p)$ 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 $r' \in SUP(P^I, I, p)$, note that $I \models Body(r)$ implies $I \models Body^+(r) = Body(r')$.

For the $\subseteq$ direction, take any $r' \in SUP(P^I, I, p)$. 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 $r \in SUP(P, I, p)$. Since $r' \in SUP(P^I, I, p)$, we get $I \models Body(r')$ but this is equivalent to $I \models Body^+(r)$. As we had $I \models Body^-(r)$, we conclude $I \models Body(r)$ and so r is supported in P given I. □

Definition 1 (*Support Graph/Explanation*)
Let P be a labelled program and I a classical model of P. A *support graph* 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 total function $\lambda : I \to Lb(P)$ assigns a label to each atom, and G further satisfies:

(i) λ is injective
(ii) for every $p \in I$, the rule r such that $Lb(r) = \lambda(p)$ satisfies:
$r \in SUP(P, I, p)$ and $B^+(r) = \{q \mid (q, p) \in E\}$.

A support graph G is said to be an *explanation* if it additionally satisfies:

(iii) G is acyclic. □

Condition (i) means that there are no repeated labels in the graph, that is, $\lambda(p) \neq \lambda(q)$ for different atoms $p, q \in I$. Condition (ii) 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). In general, program P may allow alternative ways of deriving an atom p in a model I. Thus, a same model I may have multiple support graphs under P, as we will illustrate later.

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 (ii) 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 $p \in I$.

https://doi.org/10.1017/S1471068424000048 Published online by Cambridge University Press

Definition 2 (*Supported/Justified model*)
A classical model I of a labelled program P if $I \models P$ is said to be a *supported model* of P if there exists some support graph of I under P. Moreover, I is said to be a *justified model* of P if there exists some explanation G (i.e., acyclic support graph) of I under P. We write $SPM(P)$ and $JM(P)$ to respectively stand for the set of supported and justified models of P. □

The name of *supported model* is not casual: we prove later on that, for non-disjunctive programs, the above definition coincides with the traditional one in terms of fixpoints of the immediate consequence operator (van Emden and Kowalski 1976) or as models of Clark's completion (Clark 1978). From Definition 2, it is clear that all justified models are obviously supported $JM(P) \subseteq SPM(P)$ but, in general, the opposite does not hold, as we will see later. Our main focus, however, is on justified models, since we will relate them to proofs, that are always acyclic. 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, so its support is always empty $SUP(P, I, c) = \emptyset$ 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 explanations:

$$\ell_1 : a \longrightarrow \ell_2 : d \qquad\qquad \ell_1 : a \qquad \ell_3 : d. \tag{2}$$

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

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

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:

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

https://doi.org/10.1017/S1471068424000048 Published online by Cambridge University Press

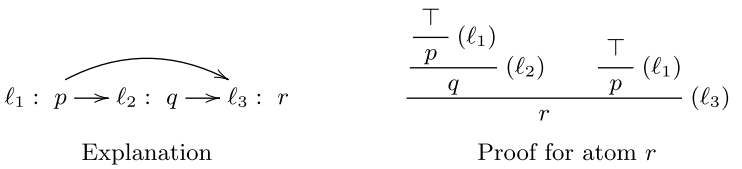

Fig. 1. Some results for model $\{p, q, r\}$ of program in Example 2.

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). □

The next proposition trivially follows from the definition of explanations:

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. To prove that stable models are justified, we start proving a correspondence between explanations for any model I of P and explanations under P^I.

Proposition 3
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 $SUP(P, I, p)$ and $SUP(P^I, I, p)$ 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 $SUP(P^I, I, p)$ are the positive parts of the rules in $SUP(P, I, p)$, so the graphs we can form are also the same. □

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

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

Proof
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:

https://doi.org/10.1017/S1471068424000048 Published online by Cambridge University Press

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. $I_i \models \mathit{Body}(r) \wedge \neg \mathit{Head}(r)$
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: $G_{i+1} \leftarrow \langle I_i, E_i, \lambda_i \rangle$
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 $p \in I \cap \mathit{Head}(r)$ (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, $I_i \models \mathit{Body}(r)$ implies $I \models \mathit{Body}(r)$, but since $I \models P^I$, we also conclude $I \models r$ and thus $I \models \mathit{Head}(r)$ 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 $I_i \models \neg \mathit{Head}(r)$, 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 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 $I_i \models \mathit{Body}(r)$), 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 $\mathit{Head}(r)$ becomes true). Last, for every $p \in I$, it is not hard to see that the (positive) rule $r \in P^I$ such that $\mathit{Lb}(r) = \lambda_n(p)$ 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 $I \models \mathit{Body}(r)$ because $I_i \models \mathit{Body}(r)$, r is a positive rule and $I_i \subseteq I$. □

As a result, we get $\mathit{SM}(P) \subseteq \mathit{JM}(P) \subseteq \mathit{SPM}(P)$, that is, justified models lay in between stable and supported.

Proposition 4
If P is a consistent Horn program, then it has a unique justified model I that coincides with the least model of P.

Proof
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

https://doi.org/10.1017/S1471068424000048 Published online by Cambridge University Press

$p \in J \setminus I$. 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. □

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[2]:

Example 3 (*A chain of firing squads*)
Consider the following variation of the classical *Firing Squad Scenario* introduced by Pearl (1999) 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 on 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

[2] This example was already introduced as Program 7.1 by Fandinno (2015) in his PhD dissertation.

https://doi.org/10.1017/S1471068424000048 Published online by Cambridge University Press

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 $SM(P) = JM(P)$. □

Proof
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 3, 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 4, we know I is also the *least model* of P^I, which makes it a stable model of P. □

Moreover, for non-disjunctive programs, we can prove that our definition of supported model, coincides with the traditional one in terms of fixpoints of the immediate consequence operator (van Emden and Kowalski 1976). Given a non-disjunctive program P, let $T_P(I)$ be defined as $\{p \mid r \in P, I \models Body(r), Head(r) = p\}$.

Theorem 3
If P is a non-disjunctive program, then $I = T_P(I)$ iff $I \in SPM(P)$. □

Proof
For left to right, suppose $I = T_P(I)$. It is easy to see that this implies $I \models P$. By definition of T_P, for each atom p there exists some rule r with $Head(r) = p$ and $I \models Body(r)$. Let us arbitrarily pick one of those rules r_p for each p. Then, we can easily form a support graph where $\lambda(p) = Lb(r_p)$ and assign all the incoming edges for p as (q, p) such that $q \in Body^+(r_p)$.

For right to left, suppose $I \models P$ and there is some support graph G of I under P. We prove both inclusion directions for $I = T_P(I)$. For $\subseteq$, suppose $p \in I$. Then p is a node in G and there is a rule r such that $\lambda(p) = Lb(r)$, $p = Head(r)$ (P is non–disjunctive) and $I \models Body(r)$. But then $p \in T_P(I)$. For $\supseteq$, take any $p \in T_P(I)$ and suppose $p \notin I$. Then, we have at least some rule $r \in P$ with $I \models Body(r)$ and $I \not\models Head(r)(= p)$, something that contradicts $I \models P$. □

To illustrate supported models in the disjunctive case, consider the program:

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

The only justified model of this program is $\emptyset$ which is also stable and supported. Yet, we also obtain a second supported model $\{b, c\}$ that is justified by the (cyclic) support graph with labelling $\{\ell_1 : b, \ell_2 : c\}$.

3 An ASP encoding to compute explanations

In this section, we focus on the computation of explanations for a given stable model. We assume that we use an ASP solver to obtain the answer sets of some program P and

https://doi.org/10.1017/S1471068424000048 Published online by Cambridge University Press

that we have some way to label the rules. For instance, we may use the code line number (or another tag specified by the user), followed by the free variables in the rule and some separator. In that way, after grounding, we get a unique identifier for each ground rule.

To explain the answer sets of P, we may build the following (non-ground) ASP program $x(P)$ that can be fed with the (reified) true atoms in I 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. The advantage of this technique is that, rather than collecting all possible explanations in a single shot, something that is too costly for explaining large programs, we can perform regular calls to an ASP solver for $x(P, I)$ 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).

For each rule in P of the form (1), $x(P)$ contains the set of rules:

$$sup(\ell) \leftarrow as(q_1) \wedge \cdots \wedge as(q_n) \wedge as(p_i) \wedge \neg as(s_1) \wedge \cdots \wedge \neg as(s_j), \tag{3}$$

$$\wedge\ \neg\neg as(t_1) \wedge \cdots \wedge \neg\neg as(t_k), \tag{4}$$

$$\{f(\ell, p_i)\} \leftarrow f(q_1) \wedge \cdots \wedge f(q_n) \wedge as(p_i) \wedge sup(\ell), \tag{5}$$

$$\bot \leftarrow f(\ell, p_i) \wedge f(\ell, p_h), \tag{6}$$

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

$$f(A) \leftarrow f(L, A) \wedge as(A), \tag{7}$$

$$\bot \leftarrow not\ f(A) \wedge as(A), \tag{8}$$

$$\bot \leftarrow f(L, A) \wedge f(L', A) \wedge L \neq L' \wedge as(A). \tag{9}$$

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:

$$\{f(\ell, p)\} \leftarrow f(q_1) \wedge \cdots \wedge f(q_n) \quad \text{for each rule } r \in P \text{ of the form of (1)}, \; I \models Body(r), p \in H(r) \cap I, \tag{10}$$

$$\bot \leftarrow f(\ell, p_i) \wedge f(\ell, p_j) \quad \text{for each rule } r \in P \text{ of the form of (1)}, \; p_i, p_j \in H(r),\ p_i \neq p_j, \tag{11}$$

$$f(a) \leftarrow f(\ell, a) \quad \text{for each } a \in I, \tag{12}$$

$$\bot \leftarrow not\ f(a) \quad \text{for each } a \in I, \tag{13}$$

$$\bot \leftarrow f(\ell, a) \wedge f(\ell', a) \quad \text{for each } a \in I,\ \ell \neq \ell'. \tag{14}$$

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

https://doi.org/10.1017/S1471068424000048 Published online by Cambridge University Press

Proof

We have to prove that J induces a valid explanation G. Let us denote $At(J) \stackrel{\text{df}}{=} \{a \in At \mid f(a) \in J\}$. Since (12) 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 (13) forcing $f(a) \in J$. On the other hand, $At(J) \subseteq I$ because the only rules with $f(a)$ in the head are (12) 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 (14) 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 (i) requires that λ is injective, something guaranteed by (11). Condition (ii) 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 (10) 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 that is, $q_i \in I$, for all atoms in the positive body of rule labelled with ℓ. Note also that (10) is such that $I \models \mathit{Body}(r)$, so the rule supports atom p under I, that is, $r \in \mathit{SUP}(P, I, p)$. Let E be the set of edges formed in this way. Condition (iii) 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 (10) have now the form:

$$f(\ell, p) \leftarrow f(q_1) \wedge \cdots \wedge f(q_n), \tag{15}$$

for each rule $r \in P$ like (1), $I \models \mathit{Body}(r)$, $p \in H(r) \cap I$ 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 (15) in $x(P, I)^J$ per each choice (10) in $x(P, I)$. Now, since J is an answer set of $x(P, I)$, by monotonicity of constraints, it satisfies (11), (13), and (14) and is an answer set of the rest of the program P' formed by rules (15) and (12). This means that J is a minimal model of P'. Suppose we have a cycle in E, formed by the (labelled) nodes and edges $(\ell_1 : p_1) \longrightarrow \ldots \longrightarrow (\ell_n : p_n) \longrightarrow (\ell_1 : p_1)$. 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 (15) or (11) not satisfied by J'. Suppose J' does not satisfy some rule (11) 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 (15), 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

https://doi.org/10.1017/S1471068424000048 Published online by Cambridge University Press

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. □

Theorem 5 (*Completeness*)
Let I be an answer set of P. For every explanation $G = \langle I, E, \lambda \rangle$ of I under P, there exists a unique answer set J of program $x(P, I)$ where $f(\ell, a) \in J$ iff $\lambda(a) = \ell$ in G. □

Proof
Let I be an answer set of P and $G = \langle I, E, \lambda \rangle$ be 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 (7) because it contains an atom $f(a)$ for each $a \in I$. We can also see that J satisfies (11) because graph G does not contain repeated labels, so we cannot have two different atoms with the same label. The third constraint (14) 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 (11) 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, (10), 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 (15) 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 (12). 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 $x(P, I)^J$. □

4 Related work

The current approach constitutes the formal basis of the new version of the explanation tool `xclingo` (Cabalar and Muñiz 2023) which also uses the ASP encoding from Section 3 to compute the explanations. Theorems 4 and 5 prove, in this way, that the tool is sound and complete with respect to the definition of explanation provided in the current paper.

There exist many other approaches for explanation and debugging in ASP (see the survey by Fandinno and Schulz (2019)). The closest approach to the current work is clearly the one based on *causal graphs* (Cabalar *et al.* 2014). Although we conjecture that a formal relation can be established (we plan this for future work), the main difference is that, in causal graphs, we get a unique, complete, and individual explanation for each atom, whereas the current work obtains the multiple explanations of a given model (a set of atoms). 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,

https://doi.org/10.1017/S1471068424000048 Published online by Cambridge University Press

we would get an individual derivation in each support graph, 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 by Erdem and Oztok (2013). There, the authors implemented an ad hoc approach to the problem of solving biomedical queries, rather than a general ASP explanation tool.

Other examples of general approaches are the *formal theory of justifications* (Denecker *et al.* 2015), *off-line justifications* (Pontelli and Son 2006), LABAS (Schulz and Toni 2016) (based on argumentation theory (Bondarenko *et al.* 1997; Dung *et al.* 2009)) or s(CASP) (Arias *et al.* 2020). All of them provide graph or 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 nested graph-based justifications (Marynissen 2022) and is actually a more general framework that allows covering other logic programming semantics. System `xASP` (Trieu *et al.* 2022) generates explanation graphs previously proposed by Pontelli and Son (2006) and also uses an ASP meta-programming encoding. In the case of s(CASP), it proceeds in a top-down manner, building the explanation as an ordered list of literals extracted from the goal-driven satisfaction of the query. An important difference with respect to this last group of approaches is that their explanations consider dependences through default negation. To illustrate the effect, take the program:

$$\begin{aligned} \ell_1 &: switch \\ \ell_2 &: light \leftarrow switch, not\ ab \\ \ell_3 &: ab \leftarrow blown\_fuse \\ \ell_4 &: ab \leftarrow broken\_bulb \\ \ell_5 &: ab \leftarrow blackout, not\ generator. \end{aligned}$$

The only stable model is $\{switch, light\}$ and its unique explanation is the support graph

$$\ell_1 : switch \longrightarrow \ell_2 : light,$$

that is, the light is on because we toggled the switch. Adding negative information would lead us to explain *not ab* and obtain two explanations: one in which we also add that there is no blown fuse, no broken bulb and no blackout; the second one is similar, but instead of no blackout, we have a doubly negative dependence on generator: that is, nothing prevents having a generator, even though we do not have it. Note how these explanations may easily get complicated: we could have to negate multiple alternative ways of breaking the bulb, even when *none of them have happened*[3]. Our approach consists, instead, in explaining the information that currently holds, assuming that other states of affairs will arise in terms of other alternative answer sets. In other words, we refrain from using facts for which we have no evidence or reason to believe in our current model.

Another distinctive feature of our approach is that it provides explanations for disjunctive programs and, moreover, it has also allowed us to define supported and justified models for that case. In fact, we plan to study potential connections between justified

[3] We face here, somehow, a kind of qualification problem in the explanations.

https://doi.org/10.1017/S1471068424000048 Published online by Cambridge University Press

models and other approaches for disjunction not based in minimal models such as in the works by Aguado *et al.* (2019) or Shen and Eiter (2019).

Other ASP explanation approaches have to do with comparing stable models or explaining their non-existence. For instance, Gebser *et al.* (2008) use a meta-programming technique to explain why a given model *is not* an answer set of a given program. More recently, Eiter *et al.* (2019) considered the explanation of ASP programs that have no answer sets in terms of the concept of *abstraction* (Saribatur *et al.* 2021). This allows spotting which parts of a given domain are actually relevant for rising the unsatisfiability of the problem. We plan to explore formal relations to these approaches or to study potential combinations with some of them.

5 Conclusions

We have introduced the notion of explanation of a model of a logic program as some kind of (acyclic) labelled graph we called *support graph*. We have defined justified models as those that have at least one explanation and proved that all stable models are justified, whereas the opposite does not hold, at least for disjunctive programs. We also provided a meta-programming encoding in ASP that generates the explanations of a given stable model. We formally proved a one-to-one correspondence between the answer sets of the encoding and the explanations of the original stable model. Since this encoding constitutes the basis of the tool `xclingo 2.0`, we provide in this way a formal proof of correctness for this system. A system description of the tool is left for a forthcoming document. Future work includes the comparison to other approaches, the explanation of unsatisfiable programs and the minimisation or even the specification of preferences among explanations.

References

Aguado, F., Cabalar, P., Fandinno, J., Pearce D., Pérez G. and Vidal C. 2019. Forgetting auxiliary atoms in forks. *Artificial Intelligence*, *275*, 575–601.

Arias, J., Carro, M., Chen, Z. and Gupta, G. 2020. Justifications for goal-directed constraint answer set programming. In *International Conference on Logic Programming (ICLP 2020)*.

Bondarenko, A., Dung, P., Kowalski, R. and Toni, F. 1997. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, *93*, 1, 63–101.

Brewka, G., Eiter, T. and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM*, *54*, 12, 92–103.

Cabalar, P., Fandinno, J. and Fink, M. 2014. Causal graph justifications of logic programs. *Theory and Practice of Logic Programming*, *14*, 603–618.

Cabalar, P. and Muñiz, B. 2023. Explanation graphs for stable models of labelled logic programs. In *Proceedings of the International Conference on Logic Programming 2023 Workshops co-located with the 39th International Conference on Logic Programming (ICLP 2023), London, United Kingdom, July 9th and 10th, 2023*, J. Arias, S. Batsakis, W. Faber, G. Gupta, F. Pacenza, E. Papadakis, L. Robaldo, K. Rückschloß, E. Salazar, Z. G. Saribatur, I. Tachmazidis, F. Weitkämper, and A. Z. Wyner, Eds., vol. 3437. CEUR Workshop Proceedings. CEUR-WS.org.

Clark, K. L. 1978. Negation as failure. In *Logic and Databases*, H. Gallaire and J. Minker, Eds. Plenum, 293–322.

https://doi.org/10.1017/S1471068424000048 Published online by Cambridge University Press

Denecker, M., Brewka, G. and Strass, H. 2015. A formal theory of justifications. In *Logic Programming and Nonmonotonic Reasoning – 13th International Conference, LPNMR 2015, Lexington, KY, USA, Proceedings*, F. Calimeri, G. Ianni, and M. Truszczynski, Eds., vol. 9345. Lecture Notes in Computer Science. Springer.

Dung, P., Kowalski, R. and Toni, F. 2009. Assumption-based argumentation. *Argumentation in Artificial Intelligence*, 199–218.

Eiter, T., Saribatur, Z. and Schüller, P. 2019. Abstraction for zooming-in to unsolvability reasons of grid-cell problems. In *International Joint Conference on Artificial Intelligence IJCAI 2019, Workshop on Explainable Artificial Intelligence*.

Erdem, E. and Oztok, U. 2013. Generating explanations for complex biomedical queries. *Theory and Practice of Logic Programming*, *15*.

Fandinno, J. 2015. *A Causal Semantics for Logic Programming*. PhD thesis, Facultad de Informática, University of A Coruña.

Fandinno, J. and Schulz, C. 2019. Answering the "why" in answer set programming – A survey of explanation approaches. *Theory and Practice of Logic Programming*, *19*, 2, 114–203.

Gebser, M., Pührer, J., Schaub, T. and Tompits, H. 2008. A meta-programming technique for debugging answer-set programs. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence, Chicago, IL, USA*, D. Fox and C. P. Gomes, Eds. AAAI Press.

Gelfond, M. and Lifschitz, V. 1988. The stable models semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming*, 1070–1080.

Marynissen, S. 2022. *Advances in Justification Theory*. PhD thesis, Department of Computer Science, KU Leuven. Denecker, Marc and Bart Bogaerts (supervisors).

Pearl, J. 1999. Reasoning with cause and effect. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence, IJCAI 1999, Stockholm, Sweden*, T. Dean, Ed. Morgan Kaufmann.

Pontelli, E. and Son, T. C. 2006. Justifications for logic programs under answer set semantics. In *Logic Programming*, S. Etalle and M. Truszczyński, Eds., Berlin, Heidelberg: Springer, 196–210.

Saribatur, Z. G., Eiter, T. and Schüller, P. 2021. Abstraction for non-ground answer set programs. *Artificial Intelligence*, *300*, 103563.

Schulz, C. and Toni, F. 2016. Justifying answer sets using argumentation. *Theory and Practice of Logic Programming*, *16*, 1, 59–110.

Shen, Y. and Eiter, T. 2019. Determining inference semantics for disjunctive logic programs. *Artificial Intelligence*, *277*.

Trieu, L. L. T., Son, T. C. and Balduccini, M. 2022. xasp: An explanation generation system for answer set programming. In *Logic Programming and Nonmonotonic Reasoning – 16th International Conference, LPNMR 2022, Genova, Italy, September 5–9, 2022, Proceedings*, G. Gottlob, D. Inclezan, and M. Maratea, Eds., vol. 13416. Lecture Notes in Computer Science. Springer, 363–369.

van Emden, M. H. and Kowalski, R. A. 1976. The semantics of predicate logic as a programming language. *Journal of the ACM*, *23*, 733–742.

https://doi.org/10.1017/S1471068424000048 Published online by Cambridge University Press