Advancements in xASP, an XAI System for Answer Set Programming

Mario Alviano[1,*], Ly Ly Trieu[2], Tran Son[2] and Marcello Balduccini[3]

[1]DEMACS, University of Calabria, Via Bucci 30/B, 87036 Rende (CS), Italy
[2]New Mexico State University, USA
[3]Saint Joseph's University, USA

Abstract

Explainable artificial intelligence (XAI) aims at addressing complex problems by coupling solutions with reasons that justify the provided answer. In the context of Answer Set Programming (ASP) the user may be interested in linking the presence or absence of an atom in an answer set to the logic rules involved in the inference of the atom. Such explanations can be given in terms of directed acyclic graphs (DAGs). This article reports on the advancements in the development of the XAI system `xASP` by revising the main foundational notions and by introducing new ASP encodings to compute minimal assumption sets, explanation sequences, and explanation DAGs.

Keywords

Answer Set Programming, eXplainable Artificial Intelligence, Knowledge Representation and Reasoning

1. Introduction

The interest in explainable artificial intelligence (XAI) has grown substantially in recent years. The reasons for this trend are obvious: while intelligent systems capable of solving complex problems are useful, confidence in their results is limited unless users can query them about the reasons that lead to the solutions produced. The *right to an explanation* law, extensively discussed in the USA, EU and UK, and partly enacted in some countries, increases the need for XAI systems. In this paper, we focus on the development of an XAI system for Answer Set Programming (ASP) [1, 2]. ASP is a knowledge representation and reasoning (KR&R) approach to problem solving using logic programs under answer set semantics [3], an extension of Datalog with a strong connection with well-founded semantics [4]. In this setting, we are mainly interested in the question "*given an answer set A of a program Π and an atom α, why does $\alpha \in A$ (or $\alpha \notin A$)?*"

As a logic program Π is a set of rules, the question can be answered by providing the subset of Π that supports the presence (or the absence) of α given Π and A. If Π is a Datalog program,

CILC'23: 38th Italian Conference on Computational Logic, June 21–23, 2023, Udine, Italy
*Corresponding author.
mario.alviano@unical.it (M. Alviano); lytrieu@nmsu.edu (L. L. Trieu); tson@cs.nmsu.edu (T. Son); mbalducc@sju.edu (M. Balduccini)
https://alviano.net/ (M. Alviano); https://www.cs.nmsu.edu/~tson/ (T. Son); https://directory.sju.edu/marcello-balduccini (M. Balduccini)
0000-0002-2052-2063 (M. Alviano); 0000-0003-1482-9453 (L. L. Trieu); 0000-0001-5445-3054 (M. Balduccini)

then its models are easily explainable by the derivation procedure implemented by Datalog engines. Essentially, each atom in the model is explained by the support provided by a rule whose body is true and contains only already explained atoms. If Π is a logic program under the well-founded semantics, then the fact that α belongs (or does not belong) to the well-founded model of Π can be explained similarly, with the addition of some atoms that are concluded to be false because belonging to some unfounded set. Generally speaking, explanations for logic programs under the answer set semantics can also be produced in a similar way *under the assumption provided by the answer sets themselves for the interpretation of false atoms*. However, taking all false atoms as an assumption would likely result in a *faint* explanation, actually in an explanation by faith for all such false atoms. Therefore, two main issues need to be tackled in explaining the assignment of α in A: (*i*) how to compute a hopefully small set of assumptions capable of explaining the assignment of α in A; and (*ii*) how to handle constraints and rules acting as constraints, which can be taken into account to explain the falsity of some atoms in easily understandable terms.

An XAI system providing the reasons for the presence or absence of a given atom in an answer set finds another important application in the identification of the cause of unexpected results. This is a feature that can be particularly useful to the designers of complex systems confronted with unexpected inferences. In fact, identifying the root causes of those inferences can be daunting due to the many possible interactions in large knowledge bases. We found ourselves faced with such a challenge during the recent development of a commercial application. The ASP program that powered the decision-making component comprised a number of modules that could be enabled or disabled depending on needs. During development, we noticed that certain combinations of modules yielded unexpected results. After carefully checking each module, individually, for errors, we began to suspect that rules from different modules were interacting with each other in unexpected ways. Investigating those interactions proved to be a very time-consuming task that took approximately 3 *Full-Time Equivalent* weeks and considerably slowed down the project at a critical time. While XAI-inspired research conducted by the ASP community had already produced a number of tools related to this problem, such as `xclingo` [5], `DiscASP` [6], `xASP` [7], and `s(CASP)` [8], none of them could be used for our problem, due to inability to process a program of the size of ours in an acceptable amount of time, to lack of support for certain advanced language features, and in some cases due to shortcomings in the type of information produced.

In this paper, we present research advancements on the XAI system `xASP` [7], culminating with the release of the second version of `xASP`, which in particular let us obtain explanations for the unexpected interactions in the commercial application mentioned above. Our main contributions are the following:

- A notion of explanation for the presence or absence of an atom in an answer set in terms of easy-to-understand inferences originating from a hopefully small set of atoms assumed false (Section 3).
- A representation of explanations in terms of directed acyclic graphs, restricted to the atoms involved in the explanation (Section 3), and a proof of existence for the explanations according to the given definition (Section 4).

- The implementation `xASP2` , a system for producing explanations powered by ASP (Section 5), and the empirical evaluation of `xASP2` on the commercial application mentioned in this introduction (Section 6).

2. Background

All sets and sequences considered in this paper are finite. Finite, possibly empty sequences of elements are denoted by over-lined symbols. Let **P**, **C**, **V** be fixed nonempty sets of *predicate names*, *constants* and *variables*. Predicates are associated with an *arity*, a non-negative integer; let $\bot$ be a fixed predicate of arity 0 in **P** (that we will enforce to be false when we will define the notion of interpretation). A *term* is any element in $\mathbf{C} \cup \mathbf{V}$. An *atom* is of the form $p(\overline{t})$, where $p \in \mathbf{P}$, and $\overline{t}$ is a possibly empty sequence of terms. A *literal* is an atom possibly preceded by the default negation symbol *not*; they are referred to as positive and negative literals.

A *rule* is of the form

$$head \leftarrow body \tag{1}$$

where *head* is an atom, and *body* is a possibly empty sequence of literals. For a rule r, let $H(r)$ denote the atom in the head of r; let $B^+(r)$ and $B^-(r)$ denote the sets of positive and negative literals in the body of r; let $B(r)$ denote the set $B^+(r) \cup B^-(r)$. If $H(r) = \bot$, r is also called *constraint*. If $B(r) = \emptyset$, r is also called *fact*. A variable X occurring in $B^+(r)$ is a *safe variable*. And any other variable occurring in r is an *unsafe variable*. A *safe rule* is a rule with no *unsafe variables*. A *program* Π is a set of safe rules.

Example 1 (Graph 3-Colorability)**.** Given an undirected graph G encoded by predicates *node*/*1* and *edge*/*2*, the following program assigns a color among *red*, *green* and *blue* to each node so that adjacent nodes have different colors:

$$assign(X, red) \leftarrow node(X),\ not\ assign(X, green),\ not\ assign(X, blue) \tag{2}$$
$$assign(X, green) \leftarrow node(X),\ not\ assign(X, red),\ not\ assign(X, blue) \tag{3}$$
$$assign(X, blue) \leftarrow node(X),\ not\ assign(X, green),\ not\ assign(X, red) \tag{4}$$
$$\bot \leftarrow edge(X, Y),\ assign(X, C),\ assign(Y, C) \tag{5}$$

Note that all variables are safe, and that (5) is a constraint. ■

A substitution σ is a partial function from variables to constants; the application of σ to an expression (i.e., term, atom, literal, or rule) E is denoted by $E\sigma$. Let $instantiate(\Pi)$ be the program obtained from rules of Π by substituting variables with constants in **C**, in all possible ways. The Herbrand base of Π, denoted $base(\Pi)$, is the set of ground atoms (i.e., atoms with no variables) occurring in $instantiate(\Pi)$ that are **different from** $\bot$.

Example 2. Let Π_{run} comprise rules (2)–(5), and facts over *node*/1 and *edge*/2 encoding the undirected graph in Figure 1. Hence, $instantiate(\Pi_{run})$ contains, among other rules,

$$assign(a, red) \leftarrow node(a),\ not\ assign(a, green),\ not\ assign(a, blue)$$
$$\bot \leftarrow edge(a, b),\ assign(a, red),\ assign(b, red)$$

and $base(\Pi_{run})$ contains $assign(a, red)$, $assign(a, green)$, $assign(a, blue)$, and so on. ■

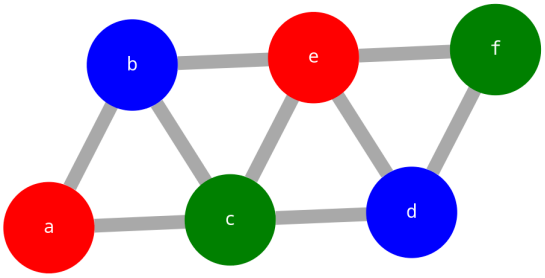

Figure 1: The undirected graph and its 3-coloring used as running example

A *(two-valued) interpretation* is a set of ground atoms **not containing** $\bot$. For a two-valued interpretation I, relation $I \models \cdot$ is defined as follows: for a ground atom $p(\overline{c})$, $I \models p(\overline{c})$ if $p(\overline{c}) \in I$, and $I \models \mathit{not}\ p(\overline{c})$ if $p(\overline{c}) \notin I$; for a ground rule r, $I \models B(r)$ if $I \models \alpha$ for all $\alpha \in B(r)$, and $I \models r$ if $I \models H(r)$ whenever $I \models B(r)$; for a program Π, $I \models \Pi$ if $I \models r$ for all $r \in \mathit{instantiate}(\Pi)$. The *reduct* of Π w.r.t. I is the program comprising the rules of $\mathit{instantiate}(\Pi)$ whose body is true w.r.t. I, that is, $\mathit{reduct}(\Pi, I) := \{r \in \mathit{instantiate}(\Pi) \mid I \models B(r)\}$. An *answer set* of Π is an interpretation A such that $A \models \Pi$ and no $I \subset A$ satisfies $I \models \mathit{reduct}(\Pi, A)$.

Example 3. Among the answer sets of program Π_{run} there is A_{run} shown in Figure 1, containing, among others, the atoms $\mathit{assign}(a, \mathit{red})$, $\mathit{assign}(b, \mathit{blue})$, and $\mathit{assign}(c, \mathit{green})$. ■

A *three-valued interpretation* is a pair (L, U), where L, U are sets of ground atoms **not containing** $\bot$ and such that $L \subseteq U$; let $lb((L,U))$ denote the lower bound L of (L,U); let $ub((L,U))$ denote the upper bound U of (L,U); hence, atoms in L are true, atoms in $U \setminus L$ are undefined, and all other atoms are false. The *evaluation function* $[\![\cdot]\!]_L^U$ associates literals with a truth value among **u**, **t** and **f** as follows: $[\![p(\overline{c})]\!]_L^U = [\![\mathit{not}\ p(\overline{c})]\!]_L^U = \mathbf{u}$ if $p(\overline{c}) \in U \setminus L$; $[\![\alpha]\!]_L^U = \mathbf{t}$ if $[\![\alpha]\!]_L^U \neq \mathbf{u}$ and $L \models \alpha$; and $[\![\alpha]\!]_L^U = \mathbf{f}$ if $[\![\alpha]\!]_L^U \neq \mathbf{u}$ and $L \not\models \alpha$. The evaluation function extends to rule bodies as follows: $[\![B(r)]\!]_L^U = \mathbf{f}$ if there is $\alpha \in B(r)$ such that $[\![\alpha]\!]_L^U = \mathbf{f}$; $[\![B(r)]\!]_L^U = \mathbf{t}$ if $[\![\alpha]\!]_L^U = \mathbf{t}$ for all $\alpha \in B(r)$; otherwise $[\![B(r)]\!]_L^U = \mathbf{u}$.

Example 4. Let $B(r) = \{\mathit{node}(a),\ \mathit{not}\ \mathit{assign}(a, \mathit{green}),\ \mathit{not}\ \mathit{assign}(a, \mathit{blue})\}$. Hence, $[\![B(r)]\!]_{\{\mathit{node}(a)\}}^{\{\mathit{node}(a),\ \mathit{assign}(a,\mathit{green}),\ \mathit{assign}(a,\mathit{blue})\}} = \mathbf{u}$, $[\![B(r)]\!]_{\{\mathit{node}(a)\}}^{\{\mathit{node}(a)\}} = \mathbf{t}$, and $[\![B(r)]\!]_{\{\mathit{node}(a),\ \mathit{assign}(a,\mathit{green})\}}^{\{\mathit{node}(a),\ \mathit{assign}(a,\mathit{green}),\ \mathit{assign}(a,\mathit{blue})\}} = \mathbf{f}$. ■

Mainstream ASP systems compute answer sets of a given program Π by applying several inference rules on (a subset of) $\mathit{instantiate}(\Pi)$, the most relevant ones for this work are summarized below. Let (L, U) be a three-valued interpretation, and $p(\overline{c})$ be a ground atom such that $[\![p(\overline{c})]\!]_L^U = \mathbf{u}$. Atom $p(\overline{c})$ in $H(r)$ is *inferred true by "support"* if $[\![B(r)]\!]_L^U = \mathbf{t}$. Atom $p(\overline{c})$ is *inferred false by "lack of support"* if each rule $r \in \mathit{instantiate}(\Pi)$ with $p(\overline{c})$ occurring in $H(r)$ is such that $[\![B(r)]\!]_L^U = \mathbf{f}$. Atom $p(\overline{c})$ is *inferred false by a "constraint-like rule"* $r \in \mathit{instantiate}(\Pi)$ if $p(\overline{c}) \in B^+(r)$, $[\![H(r)]\!]_L^U = \mathbf{f}$ and $[\![B(r) \setminus \{p(\overline{c})\}]\!]_L^U = \mathbf{t}$. Atom $p(\overline{c})$ is *inferred false by "well-founded computation"* if it belongs to some *unfounded set* X for Π w.r.t. (L, U), that is, a set X such that for all rules $r \in \mathit{instantiate}(\Pi)$ at least one of the following conditions holds: (i) $H(r) \notin X$; (ii) $[\![B(r)]\!]_L^U = \mathbf{f}$; (iii) $B^+(r) \cap X \neq \emptyset$.

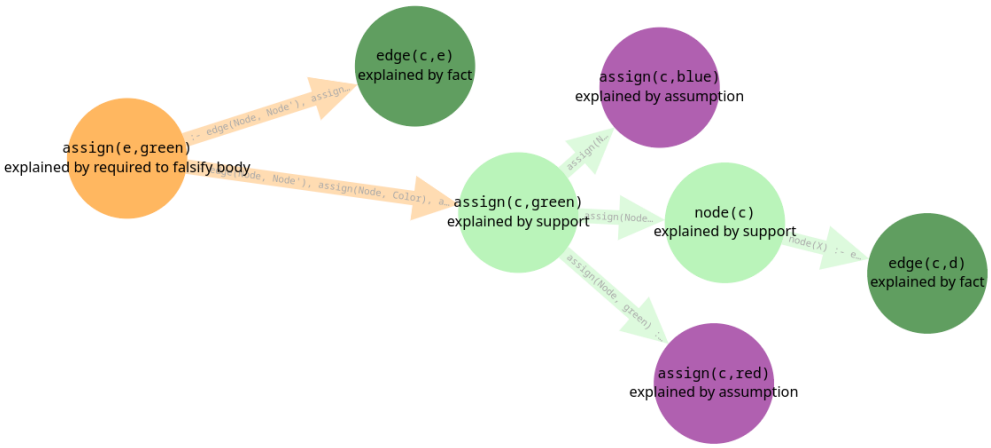

Figure 2: Induced DAG on the vertices reachable from $assign(e, green)$ for the minimal assumption set $\{assign(c, red), assign(c, blue), assign(e, red)\}$ for Π_{run}. Note that the assumption $assign(e, red)$ is not used in the portion of the DAG explaining $assign(e, green)$.

Example 5. Given $instantiate(\Pi_{run})$, and the three-valued interpretation $(\emptyset, base(\Pi_{run}))$, atom $assign(a, a)$ is inferred false by "lack of support" (or also by "well-founded computation" as $\{assign(a, a)\}$ is an unfounded set), and atom $node(a)$ is inferred true by "support". Given the three-valued interpretation $\big(\{edge(a, b),\ assign(a, red)\}, base(\Pi_{run})\big)$, atom $assign(b, red)$ is inferred false by the "constraint-like rule" (5) with substitution $\{X \mapsto a,\ Y \mapsto b,\ C \mapsto red\}$. ■

3. Explanations

In order to have easy-to-understand explanations, well-founded computation is applied only as a preprocessing step. In fact, the notion of unfounded set relies on the state of several rules in the program in input. Therefore, providing an explanation in which the user is asked to trace down such states over several steps of computation would not fulfill our requirements.

Let Π be a program, and A be one of its answer sets. A *well-founded derivation* for Π w.r.t. A, denoted $wf(\Pi, A)$, is obtained from the interpretation $(\emptyset, base(\Pi))$ by iteratively (i) adding to its lower bound atoms of A that are inferred true by support, and (ii) removing from its upper bound atoms belonging to some unfounded set.

Example 6. Given Π_{run} and A_{run} from Examples 2–3, the lower bound of $wf(\Pi_{run}, A_{run})$ contains atoms occurring as facts (i.e. rules with empty bodies). The upper bound additionally contains $assign(x, k)$ for all $x \in \{a, b, c, d, e, f\}$ and $k \in \{red, green, blue\}$. ■

An *explaining derivation* for Π and A from (L, U) is obtained by iteratively (i) adding to L atoms of A that are inferred true by support, and (ii) removing from U atoms that are inferred false by lack of support, and constraint-like rules. An *assumption set* for Π and A is a set $X \subseteq base(\Pi) \setminus A$ of ground atoms such that the explaining derivation for Π and A from $(\emptyset, ub(wf(\Pi, A)) \setminus X)$ terminates with A; in words, A is reconstructed from the false atoms of the well-founded derivation extended with X; note that $ub(wf(\Pi, A))$ is the upper bound of the three-valued interpretation $wf(\Pi, A)$. Let $AS(\Pi, A)$ be the set of assumption sets for Π

and A. A *minimal assumption set* for Π, A and a ground atom α is a set $X \in AS(\Pi, A)$ such that $X' \subset X$ implies $X' \notin AS(\Pi, A)$, and $\alpha \in X$ implies $\alpha \in X'$ for all $X' \in AS(\Pi, A)$; essentially, subset-minimal assumption sets not containing α are preferred. Let $MAS(\Pi, A, \alpha)$ be the set of minimal assumption sets for Π, A and α.

Example 7. Set $base(\Pi_{run}) \setminus A_{run}$ is an assumption set for Π_{run} and its answer set A_{run}; it is essentially the assumption underlying the definition of answer set, that is, the model can be reconstructed assuming the falsity of false atoms.

It can be checked that also $\{assign(e, red),\ assign(f, red),\ assign(f, blue)\}$ belongs to $AS(\Pi_{run}, A_{run})$. Indeed, assuming that node f is not *red* or *blue*, implies that f is *green*. The latter, combined with the assumption that node e is not *red*, implies that that e is *blue*. At this point all other colors are inferred without the need for other assumptions. ■

Given an assumption set X and an explaining derivation from $(\emptyset, ub(wf(\Pi, A)) \setminus X)$, a directed acyclic graph (DAG) can be obtained as follows: The vertices of the graph are the atoms in $base(\Pi)$. (The vertex $p(\overline{c})$ is also referred to as *not* $p(\overline{c})$.) Atoms inferred true by support due to a rule $r \in instantiate(\Pi)$ are linked to elements of $B(r)$. Any atom α inferred false by lack of support is linked to an element of $B(r)$ that is inferred false before α, for each rule $r \in instantiate(\Pi)$ such that α occurs in $H(r)$. Any atom α inferred false by a constraint-like rule $r \in instantiate(\Pi)$ is linked to the atoms occurring in $H(r)$ and the elements of $B(r) \setminus \{\alpha\}$.

Example 8. A portion of an example DAG relying on the minimal assumption set $\{assign(c, red),\ assign(c, blue),\ assign(e, red)\}$ for Π_{run} is reported in Figure 2. It can be read as follows: Atom $assign(e, green)$ is false so to falsify the body of

$$\bot \leftarrow edge(c, e),\ assign(c, green),\ assign(e, green)$$

Indeed, $edge(c, e)$ is a fact, and $assign(c, green)$ is supported by

$$assign(c, green) \leftarrow node(c),\ not\ assign(c, red),\ not\ assign(c, blue)$$

because $node(c)$ is a fact, and $assign(c, red)$ and $assign(c, blue)$ are assumed to be false. ■

4. Existence of Minimal Assumption Sets

This section is devoted to formally show that the existence of minimal assumption sets is guaranteed, and so are DAGs as defined in Section 3.

Theorem 1 (Main Theorem)**.** *Let Π be a program, A one of its answer sets, and α a ground atom in $base(\Pi)$. Set $MAS(\Pi, A, \alpha)$ is nonempty.*

In order to show the above theorem we introduce some additional notation and claims. Let Π be a program, and (L, U) be a three-valued interpretation. We denote by $\Pi, L, U \vdash \alpha$ the fact that $\alpha \in base(\Pi)$ is inferred true by support, which is the case when $[\![\alpha]\!]_L^U = \mathbf{u}$, and there is $r \in instantiate(\Pi)$ such that α occurs in $H(r)$ and $[\![B(r)]\!]_L^U = \mathbf{t}$, as defined in Section 2. Similarly, we denote by $\Pi, L, U \vdash not\ \alpha$ the fact that $\alpha \in base(\Pi)$ is inferred false by lack of support and

constraint-like rules, which is the case when $[\![\alpha]\!]_L^U = \mathbf{u}$, and one of the following conditions holds: each rule $r \in \mathit{instantiate}(\Pi)$ with α occurring in $H(r)$ is such that $[\![B(r)]\!]_L^U = \mathbf{f}$; there is $r \in \mathit{instantiate}(\Pi)$ with $\alpha \in B^+(r)$, $[\![H(r)]\!]_L^U = \mathbf{f}$ and $[\![B(r) \setminus \{\alpha\}]\!]_L^U = \mathbf{t}$.

The *explaining derivation operator* D_Π is defined as

$$D_\Pi(L,U) := (L \cup \{\alpha \in \mathit{base}(\Pi) \mid \Pi, L, U \vdash \alpha\}, U \setminus \{\alpha \in \mathit{base}(\Pi) \mid \Pi, L, U \vdash \mathit{not}\ \alpha\}).$$

Let $(L,U) \sqsubseteq (L',U')$ denote the fact that $L \subseteq L' \subseteq U' \subseteq U$, i.e., everything that is true w.r.t. (L,U) is true w.r.t. (L',U'), and everything that is false w.r.t. (L,U) is false w.r.t. (L',U').

Lemma 1. *Operator D_Π is monotonic w.r.t. $\sqsubseteq$.*

Lemma 2. *$L \subseteq A \subseteq U$ implies $lb(D_\Pi(L,U)) \subseteq A \subseteq ub(D_\Pi(L,U))$.*

The explaining derivation from (L,U) is obtained as the fix point of the sequence $(L_0,U_0) := (L,U)$, $(L_{i+1},U_{i+1}) := D_\Pi(L_i,U_i)$ for $i \geq 0$. Note that the fix point is reached in at most $|\mathit{base}(\Pi)|$ steps because of Lemma 1 and each application of D_Π reduces the undefined atoms (or is a fix point).

Lemma 3. *For any answer set A of Π, set $\mathit{base}(\Pi) \setminus A$ is an assumption set for Π and A.*

Lemma 3 essentially uses the definition of answer set (it is the minimal model of the associated program reduct) to show that there is at least one assumption set. If such an assumption set is not minimal, given the monotonicity of D_Π (Lemma 1) and the fact that D_Π does not contradict the given answer set (Lemma 2), there must exist a smaller assumption set, and eventually a subset-minimal one (as stated by the Main Theorem).

5. Generation via Meta-Programming

The concepts introduced in Section 3 can be computed by taking advantage of ASP systems. A meta-programming approach is presented in this section, where the full language of ASP is used, including constructs omitted in the previous sections, like choice rules, aggregates, weak constraints, uninterpreted functions, conditional literals and @-terms. The reader is referred to [9] for details. We will use the name *ASP programs* for encodings using the full language of ASP, in contrast to the name *program* that we use for encodings using the restricted syntax introduced in Section 2.

Program Π, answer set A and the atom to explain are encoded by a set of facts obtained by computing the unique answer set of the ASP program $\mathit{serialize}(\Pi, A, \alpha)$, defined next. Each atom $p(\overline{c})$ in $\mathit{base}(\Pi)$ is encoded by a fact `atom(`$p(\overline{c})$`)`; moreover, the encoding includes a fact `true(`$p(\overline{c})$`)` if $p(\overline{c}) \in A$, and `false(`$p(\overline{c})$`)` otherwise; additionally, if $p(\overline{c})$ is false in $\mathit{wf}(\Pi, A)$, the encoding includes a fact `explained_by(`$p(\overline{c})$`, initial_well_founded)`. As for α, the encoding includes a fact `explain(`α`)`. Each rule r of $\mathit{instantiate}(\Pi)$ is encoded by

`rule(`$\mathit{id}(\overline{X})$`) :- atom(`$p_1(\overline{t_1})$`), ..., atom(`$p_n(\overline{t_n})$`).`

where id is an identifier for r, $\overline{X}$ are the variables of r, and $B^+(r) = \{p_i(\overline{t_i}) \mid i = 1,\ldots,n\}$; moreover, the encoding includes

`head(`$id(\overline{X})$`,`$p(\bar{t})$`) :- rule(`$id(\overline{X})$`).`
`pos_body(`$id(\overline{X})$`,`$p'(\overline{t'})$`) :- rule(`$id(\overline{X})$`).`
`neg_body(`$id(\overline{X})$`,`$p''(\overline{t''})$`) :- rule(`$id(\overline{X})$`).`

for each $p(\bar{t})$ occurring in $H(r)$, $p'(\overline{t'}) \in B^+(r)$ and $p''(\overline{t''}) \in B^-(r)$.

Example 9. Recall Π_{run} and A_{run} from Examples 2–3. The ASP program $\mathit{serialize}(\Pi_{run}, A, \mathit{assign}(e, \mathit{green}))$ includes

```
atom(assign(e,green)).  false(assign(e,green)).
atom(assign(c,green)).  true(assign(c,green)).
explain(assign(e,green)).

rule(r4(X,Y,C)) :- atom(edge(X,Y)), atom(assign(X,C)), atom(assign(Y,C)).
pos_body(r4(X,Y,C), edge(X,Y))   :- rule(r4(X,Y,C)).
pos_body(r4(X,Y,C), assign(X,C)) :- rule(r4(X,Y,C)).
pos_body(r4(X,Y,C), assign(Y,C)) :- rule(r4(X,Y,C)).
```

and several other rules. The answer set of $\mathit{serialize}(\Pi_{run}, A, \mathit{assign}(a, \mathit{green}))$ includes, among other atoms, `rule(r4(c,e,green))`. ■

The ASP program Π_{MAS} reported in Figure 3, coupled with a fact for each atom in the answer set of $\mathit{serialize}(\Pi, A, \alpha)$, has optimal answer sets corresponding to cardinality-minimal elements in $\mathit{MAS}(\Pi, A, \alpha)$. Intuitively, line 1 guesses the assumption set, line 2–3 minimizes the size of the assumption set (preferring to not assume the falsity of the atom to explain), and lines 4–5 impose that each atom must have exactly one explanation. The other rules encode the explaining derivation for Π and A from $\mathit{wf}(\Pi, A) \setminus X$, where X is the guessed assumption set.

Given a minimal assumption set encoded by predicate `assume_false/1`, an explaining derivation can be computed by removing lines 1–3 from the ASP program Π_{MAS}. Let Π_{EXP} be such an ASP program. Finally, given an explaining derivation encoded by `explained_by(Index,Atom,Reason)`, with the additional `Index` argument encoding the order in the sequence, a DAG linking atoms according to the derivation can be computed by the ASP program Π_{DAG} reported in Figure 4.

Example 10. Let Π_S have a fact for each atom in the answer set of $\mathit{serialize}(\Pi_{run}, A_{run}, \mathit{assign}(e, \mathit{green}))$. It can be checked that the minimal assumption set $\{\mathit{assign}(c, \mathit{red}), \mathit{assign}(c, \mathit{blue}), \mathit{assign}(e, \mathit{red})\}$ can be generated by $\Pi_{MAS} \cup \Pi_S$.

Program $\Pi_{EXP} \cup \Pi_S \cup \{\mathit{assign}(c, \mathit{red}), \mathit{assign}(c, \mathit{blue}), \mathit{assign}(e, \mathit{red})\}$ generates an explaining derivation, for example one including the following atoms:

```
explained_by(assign(e,green), required_to_falsify_body, r4(c,e,green))
explained_by(assign(c,green), support, r2(c))
```

Let Π_E have a fact for each instance of *explained_by/3* in the explaining derivation. $\Pi_{DAG} \cup \Pi_S \cup \Pi_E$ generates a DAG, for example the one shown in Figure 2 including $\mathit{link}(\mathit{assign}(e, \mathit{green}), \mathit{assign}(c, \mathit{green}))$. ■

```
{assume_false(Atom)} :- false(Atom).
:~ false(Atom), assume_false(Atom), not explain(Atom). [1@1, Atom]
:~ false(Atom), assume_false(Atom), explain(Atom). [1@2, Atom]

has_explanation(Atom) :- explained_by(Atom,_).
:- atom(X), #count{Reason: explained_by(Atom,Reason)} != 1.

explained_by(Atom, assumption) :- assume_false(Atom).

{explained_by(Atom, (support, Rule))} :- head(Rule,Atom), true(Atom);
   true(BAtom) : pos_body(Rule,BAtom);
   has_explanation(BAtom) : pos_body(Rule,BAtom);
   false(BAtom) : neg_body(Rule,BAtom);
   has_explanation(BAtom) : neg_body(Rule,BAtom).

{explained_by(Atom, lack_of_support)} :- false(Atom);
   false_body(Rule) : head(Rule,Atom).
false_body(Rule) :- rule(Rule);
   pos_body(Rule,BAtom), false(BAtom), has_explanation(BAtom).
false_body(Rule) :- rule(Rule);
   neg_body(Rule,BAtom), true(BAtom), has_explanation(BAtom).

{explained_by(Atom, (required_to_falsify_body, Rule))} :- false(Atom);
   pos_body(Rule,Atom), false_head(Rule);
   true(BAtom) : pos_body(Rule,BAtom), BAtom != Atom;
   has_explanation(BAtom) : pos_body(Rule,BAtom), BAtom != Atom;
   false(BAtom) : neg_body(Rule,BAtom);
   has_explanation(BAtom) : neg_body(Rule,BAtom).
false_head(Rule) :- rule(Rule); false(HAtom) : head(Rule,HAtom);
   has_explanation(HAtom) : head(Rule,HAtom).
```

Figure 3: ASP program Π_{MAS} for computing a minimal assumption set

6. Implementation and Experiment

We implemented `xASP2` , an XAI system for ASP powered by the `clingo python api` [10]. It takes as input an ASP program Π, one of its answer sets A, and an atom α, and can produce in output minimal assumption sets, explaining derivations, and DAGs to help the user figure out the assignment of α. The source code is available at https://github.com/alviano/xasp and an example DAG is given at https://xasp-navigator.netlify.app/.

The pipeline implemented by `xASP2` starts with the serialization of the input data, which is obtained by means of an ASP program crafted from the abstract syntax tree of Π and whose answer set identifies the relevant portion of *instantiate*(Π) and *base*(Π). In a nutshell, ground atoms provided by the user, $A \cup \{\alpha\}$, are part of *base*(Π) and used to instantiate rules of Π (by matching positive body literals), which in turn may extend *base*(Π) with other ground atoms occurring in the instantiated rules; possibly, some atoms of *base*(Π) of particular interest can be explicitly provided by the user. The system takes care of the computation of false atoms in

```
1 link(Atom, BAtom) :- explained_by(_, Atom, (support, Rule));
2    pos_body(Rule, BAtom).
3 link(Atom, BAtom) :- explained_by(_, Atom, (support, Rule));
4    neg_body(Rule, BAtom).

5 {link(Atom, A) : pos_body(Rule,A), false(A), explained_by(I,A,_), I < Index;
6  link(Atom, A) : neg_body(Rule,A), true (A), explained_by(I,A,_), I < Index}
       = 1 :- explained_by(_, Atom, lack_of_support); head(Rule, Atom).

7 link(Atom,A) :- explained_by(_, Atom, (required_to_falsify_body, Rule));
8    head(Rule,A).
9 link(At,A) :- explained_by(_,At,(required_to_falsify_body, Rule));
10   pos_body(Rule,A), A != At.
11 link(Atom,A) :- explained_by(_,Atom,(required_to_falsify_body, Rule));
12   neg_body(Rule,A).
```

Figure 4: ASP program Π_{DAG} for computing a directed acyclic graph associated with an explaining derivation

the well-founded derivation $wf(\Pi, A)$.

Obtained $serialize(\Pi, A, \alpha)$, `xASP2` proceeds essentially as described in Section 5, by computing a minimal assumption set, an explaining derivation and an explanation DAG. As an additional optimization, the explaining derivation is shrunk to the atoms reachable from α, again by means of an ASP program. Finally, the user can opt for a few additional steps: obtain a graphical representation by means of the `igraph` network analysis package (https://igraph.org/); obtain an interactive representation in https://xasp-navigator.netlify.app/; ask for different minimal assumption sets, explaining derivations and DAGs.

We assessed `xASP2` empirically on the commercial application mentioned in the introduction. The ASP program comprises 420 rules and 651 facts. After grounding, there are 4261 ground rules and 4468 ground atoms. The program was expected to have a unique answer set, but two answer sets were actually computed. Our experiment was run on an Intel Core i7-1165G7 @2.80 GHz and 16 GB of RAM. `xASP2` computed a DAG for the unexpected true atom, `behaves_inertially(testing_posTestNeg,121)`, in 14.85 seconds on average, over 10 executions. The DAG comprises 87 links, 45 internal nodes and 20 leaves, only one of which being explained by assumption; only 30 of the 420 symbolic rules and 11 of the 651 facts are involved in the DAG; at the ground level, only 48 of the 4261 ground rules and 65 of the 4468 ground atoms are involved. Additionally, we repeated the experiment on 10 randomly selected atoms with respect to two different answer sets, repeating each test case 10 times. We measured an average runtime of 14.79 seconds, with a variance of 0.004 seconds.

7. Related Work

As mentioned in the introduction, our work is in the context of XAI, which in turn can be applied to debug by identifying a set of rules that justifies the derivation of a given atom. For example, if an atom α is supposed to be false in all answer sets of a program Π but appears

in some answer set A, an explanation graph of α could help to understand which rules are behaving anomalously. Therefore, in this section we consider some debugging tools for ASP, as well as state-of-the-art XAI systems for ASP. Table 1 reports a summary of the compared features: whether the explanation is guaranteed to be acyclic; whether the input program may include aggregates and constraints; whether the query atom can be false in the answer set; and whether the system is available for experimentation.

`xclingo` [5] can generate derivation trees for an atom in an ASP computation. Derivation trees are obtained by adding to the input program `trace_rule` and `trace` annotations, which are then compiled into theory atoms and auxiliary predicates. Then, the explanations are obtained by decoding the answer sets of the modified program. The main drawback of `xclingo` is its inability to include negative literals in the explanations and to deal with some linguistic constructs such as constraint. Due to the simplification, explanations provided by `xclingo` are not faithful to the original program. Our system overcomes such limitations.

`s(CASP)` [8] leverages top-down Prolog computation to generate a justification tree in natural language for Constraint Answer Set programs. Due to Prolog computation, different justifications are produced when the order of atoms in rules or the order of rules in the program is changed [7]. Note that our explanation graphs are not affected by program reordering.

`Visual-DLV` [11] is a GUI for developing and testing `DLV` programs, which in particular provides a command to examine why an atom is true in the latest computed answer set. Such a question is answered by providing the reason that led the solver infer the atom, among them the possibility that the atom is a branching literal (a literal guessed to be true by the backtracking algorithm). Differently from the approach proposed in this paper, in `Visual-DLV` the link with the original program is weak due to several simplifications implemented by the grounder and the solver. Moreover, while our approach minimizes the atoms whose truth value must be assumed, `Visual-DLV` by design does nothing to simplify the amount of data shown to the user to explain the derivation of an atom. For example, the answer set $\{b\}$ of $\Pi_{rw} = \{\ a \leftarrow \mathit{not}\ b.\ \ a \leftarrow b, c.\ \ b \leftarrow \mathit{not}\ a.\ \ c \leftarrow a, b.\}$ may be obtained by branching on *not c*, inferring nothing, and then on *not a*, inferring b. Asking for why c is false, would result in the

Table 1
Summary of compared features

System (if any) and reference	Acyclic explanation	Linguistic extentions	Explanation for false atoms	System availability
`xclingo` [5]	Yes	None	No	Yes
`s(CASP)` [8]	Yes	Constraints	Yes	Yes
`Visual-DLV` [11]	Yes	Constraints	No	Yes
`spock` [12]	Yes	Constraints	No	Yes
`DWASP` [13]	Yes	Constraints	No	Yes
[14]	No	None	Yes	No
[15]	Yes	None	Yes	No
`ASPeRiX` [16]	Yes	Constraints	Yes	Yes
`LABAS` [17]	No	None	Yes	Yes
[18]	No	Aggregates	Yes	No
`xASP2`	Yes	Constraints	Yes	Yes

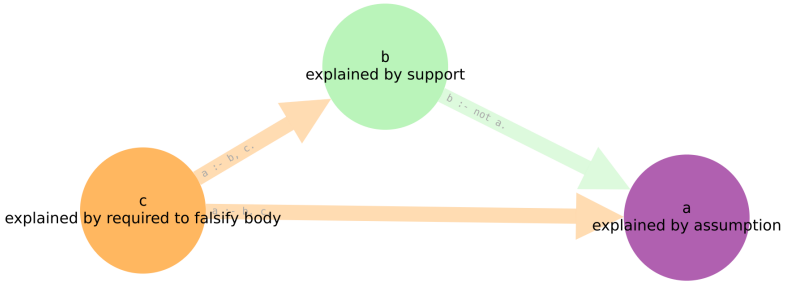

Figure 5: Induced DAG on the vertices reachable from c for the MAS $\{a\}$ of Π_{rw} (Section 7)

answer "because *not c* is a branching literal." `xASP2` assumes the falsity of a, from which the truth of b and the falsity of c can be inferred.

`spock` [12] makes use of tagging techniques [19] to translate the input program into a new program whose answer sets can be used to debug the original program. The information reported to the user includes rules whose body is true (applicable rules), rules whose body is false (blocked rules), and abnormality tags associated with completion and loop formulas. For Π_{rw} and the answer set $\{b\}$, `spock` detects the fact that the third rule is applicable, and that the other rules are blocked; the exact reason for which a rule is blocked is not reported. Within this respect, our approach is simpler and focuses on easy-to-understand inference rules that can be clearly visualized via a DAG like the one in Figure 5.

`DWASP` [13] is aimed at identifying a set of rules that are responsible for the absence of an expected answer set. It combines the grounder `gringo` [20] and an extension of the ASP solver `WASP` [21], and introduces the `gringo-wrapper` to "disable" some grounding simplifications. The expected, absent answer set is encoded as a set of constraints, so that its combination with the input program has no answer set at all, and minimal unsatisfiable subsets (MUSes) can be computed. Some questions are asked to the user so to select one MUS that makes more sense to investigate for the absence of the answer set; in fact, at that point the user has a hopefully small set of rules to investigate for bugs.

Explanation graphs can be given in terms of off-line justifications [14, 15], possibly containing cycles among false atoms [14]. For example, given $\Pi_f = \{\ a \leftarrow b.\ \ b \leftarrow a.\ \}$ and the answer set $\emptyset$, [14] explains the falsity of a by a cycle between a and b; `xASP2` and [15], instead, use the fact that a is false in the well-founded model of Π_f. We also observe that [15] fixes the assumption set to the false atoms that are left undefined by the well-founded model. On-line justifications are produced by `ASPeRiX` [16], which implements a search procedure based on the selection of rules rather than literals. In this case the explanation is produced while searching an answer set, and it is not possible to specify an answer set of interest. Other approaches relying on justifications and resulting in possibly cyclic explanation graphs are based on *assumption-based argumentation*, like `LABAS` [17], or on *trees of systems*, as proposed in [18]. Interestingly, [18] deals with aggregates, which we plan to address in our future work, even if a system implementing the approach of [18] is not discussed or released.

Finally, comparing `xASP2` with the previous version of `xASP`, we observe that the system was completely redesigned by replacing several algorithms implemented in procedural programming languages and Prolog with more declarative meta-encoding programming powered

by mainstream ASP engines. Moreover, the explanation DAGs produced by `xASP2` can be visualized in an interactive web user interface that we expect to describe in future publications.

8. Conclusion

We formalized and implemented a system for XAI targeting the ASP language and powered by ASP engines. The presence or absence of an atom in an answer set is explained in terms of easy-to-understand inferences originating from a hopefully small set of atoms assumed false. The explanation is shown as a DAG rooted at the atom to be explained, and can be computed in a few seconds in our test cases. The extension of our approach to other linguistic constructs of ASP beyond those supported by the current approach, as for example aggregates and choice rules, constitutes an interesting line of future research.

Acknowledgments

Portions of this publication and research effort are made possible through the help and support of NIST via cooperative agreement 70NANB21H167. Tran Cao Son was also partially supported by NSF awards #1757207 and #1914635. Mario Alviano was also partially supported by Italian Ministry of Research (MUR) under PNRR project FAIR "Future AI Research", CUP H23C22000860006, under PNRR project Tech4You "Technologies for climate change adaptation and quality of life improvement", CUP H23C22000370006, and under PNRR project SERICS "SEcurity and RIghts in the CyberSpace", CUP H73C22000880001; by Italian Ministry of Health (MSAL) under POS project RADIOAMICA, CUP H53C22000650006; by the LAIA lab (part of the SILA labs) and by GNCS-INdAM.

References

[1] V. Marek, M. Truszczyński, Stable models and an alternative logic programming paradigm, in: The Logic Programming Paradigm: a 25-year Perspective, 1999, pp. 375–398. doi:10.1007/978-3-642-60085-2_17.

[2] I. Niemelä, Logic programming with stable model semantics as a constraint programming paradigm, Annals of Mathematics and Artificial Intelligence 25 (1999) 241–273. doi:10.1023/A:1018930122475.

[3] M. Gelfond, V. Lifschitz, Logic programs with classical negation, in: D. Warren, P. Szeredi (Eds.), Logic Programming: Proc. of the Seventh International Conference, 1990, pp. 579–597.

[4] N. Pelov, M. Denecker, M. Bruynooghe, Well-founded and stable semantics of logic programs with aggregates, Theory Pract. Log. Program. 7 (2007) 301–353.

[5] P. Cabalar, J. Fandinno, B. Muñiz, A system for explainable answer set programming, Electronic Proceedings in Theoretical Computer Science 325 (2020) 124–136.

[6] F. Li, H. Wang, K. Basu, E. Salazar, G. Gupta, Discasp: A graph-based asp system for finding relevant consistent concepts with applications to conversational socialbots, arXiv preprint arXiv:2109.08297 (2021).

[7] L. L. Trieu, T. C. Son, M. Balduccini, xasp: An explanation generation system for answer set programming, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2022, pp. 363–369.
[8] J. Arias, M. Carro, Z. Chen, G. Gupta, Justifications for goal-directed constraint answer set programming, Electronic Proceedings in Theoretical Computer Science 325 (2020) 59–72.
[9] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, T. Schaub, Asp-core-2 input language format, Theory Pract. Log. Program. 20 (2020) 294–309.
[10] R. Kaminski, J. Romero, T. Schaub, P. Wanko, How to build your own asp-based system?!, Theory and Practice of Logic Programming 23 (2023) 299–361. doi:10.1017/S1471068421000508.
[11] S. Perri, F. Ricca, G. Terracina, D. Cianni, P. Veltri, An integrated graphic tool for developing and testing dlv programs, in: Proceedings of the Workshop on Software Engineering for Answer Set Programming (SEA'07), 2007, pp. 86–100.
[12] M. Brain, M. Gebser, J. Pührer, T. Schaub, H. Tompits, S. Woltran, That is illogical captain! the debugging support tool spock for answer-set programs: system description, in: Proceedings of the Workshop on Software Engineering for Answer Set Programming (SEA'07), 2007, pp. 71–85.
[13] C. Dodaro, P. Gasteiger, K. Reale, F. Ricca, K. Schekotihin, Debugging non-ground asp programs: Technique and graphical tools, Theory and Practice of Logic Programming 19 (2019) 290–316.
[14] E. Pontelli, T. C. Son, O. Elkhatib, Justifications for logic programs under answer set semantics, Theory and Practice of Logic Programming 9 (2009) 1–56.
[15] C. Viegas Damásio, A. Analyti, G. Antoniou, Justifications for logic programming, in: Logic Programming and Nonmonotonic Reasoning: 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proc. 12, Springer, 2013, pp. 530–542.
[16] C. Béatrix, C. Lefèvre, L. Garcia, I. Stéphan, Justifications and blocking sets in a rule-based answer set computation, in: Technical Communications of the 32nd International Conference on Logic Programming (ICLP 2016), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
[17] C. Schulz, F. Toni, Justifying answer sets using argumentation, Theory and Practice of Logic Programming 16 (2016) 59–110.
[18] S. Marynissen, J. Heyninck, B. Bogaerts, M. Denecker, On nested justification systems (full version), arXiv preprint arXiv:2205.04541 (2022).
[19] J. P. Delgrande, T. Schaub, H. Tompits, A framework for compiling preferences in logic programs, Theory and Practice of Logic Programming 3 (2003) 129–187.
[20] M. Gebser, R. Kaminski, A. König, T. Schaub, Advances in gringo series 3, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2011, pp. 345–351.
[21] M. Alviano, C. Dodaro, N. Leone, F. Ricca, Advances in wasp, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2015, pp. 40–54.