An XAI View on Explainable ASP: Methods, Systems, and Perspectives

Thomas Eiter , **Tobias Geibinger** and **Zeynep G. Saribatur**
Institute of Logic and Computation, TU Wien, Austria
{thomas.eiter, tobias.geibinger, zeynep.saribatur}@tuwien.ac.at

Abstract

Answer Set Programming (ASP) is a popular declarative reasoning and problem solving approach in symbolic AI. Its rule-based formalism makes it inherently attractive for explainable and interpretive reasoning, which is gaining importance with the surge of Explainable AI (XAI). A number of explanation approaches and tools for ASP have been developed, which often tackle specific explanatory settings and may not cover all scenarios that ASP users encounter. In this survey, we provide, guided by an XAI perspective, an overview of types of ASP explanations in connection with user questions for explanation, and describe their coverage by current theory and tools. Furthermore, we pinpoint gaps in existing ASP explanations approaches and identify research directions for future work.

1 Introduction

Understanding the decision-making of AI systems is crucial for accepting them as tools in industry and society to ease our everyday life. This need especially occurs for AI systems based on machine learning, as they usually provide little-to-no insights on their underlying models built from observed data. The field of Explainable AI (XAI) tackles the challenge of AI systems providing explanations on their decisions and their models, to help in the understanding of their mechanism [Calegari *et al.*, 2020; Schwalbe and Finzel, 2024]. Symbolic AI, which is based on describing the knowledge of the world and a problem at hand through logical or related formal languages, is in a strong position for XAI as symbolic and especially rule-based representations are more understandable to humans than what subsymbolic AI systems (largely neural networks) provide, and thus allow for transparency.

One of the core logic-based formalisms in symbolic AI is Answer Set Programming (ASP) [Brewka *et al.*, 2011; Lifschitz, 2019], which is based on logic programs under stable model semantics. In ASP, a problem is represented by a set of rules of the form $H \leftarrow B$ in a first-order language (called "program"), where intuitively, H must hold whenever B is true. The models of such programs (called answer sets [Gelfond and Lifschitz, 1991]) then correspond to the solutions of the problem. The representation often follows a "guess-and-check" methodology, where solution candidates are generated (usually via choice rules) and invalid candidates ruled out through constraints. A notable feature of ASP is nonmonotonicity, i.e., adding rules to a program may cause new answer sets and that inferences from all answer sets no longer hold. It embraces the *closed world assumption* and can be fruitfully exploited to model exceptions and defaults, e.g., inertia rules in planning.

Thanks to its declarative and expressive language, the rich support for features such as aggregates, optimisation, and preference handling, and by the availability of efficient solvers, ASP is highly popular and widely used beyond AI in Computer Science and other fields. It has been used to solve a variety of problems such as combinatorial optimisation, logical reasoning, planning, bioinformatics, and data integration to name a few [Schaub and Woltran, 2018], with growing use in industry [Rajaratnam *et al.*, 2023]. Moreover, recent work shows the potential of ASP for neuro-symbolic AI [Rader and Russo, 2025; Shakarian *et al.*, 2023] and aiding Large Language Models in reasoning [Kareem *et al.*, 2024; Rajasekharan *et al.*, 2023].

Given that an ASP program is a declarative specification without explicit control flow, users might have issues following the underlying reasoning mechanisms, i.e., why a certain result was returned by the solver. Several tools and theories for explainability in ASP exist and continue to be investigated. Those explanation concepts not only differ in what kind of answer they provide, but also in what kind of setting they operate in. A previous landmark survey on explanations in ASP [Fandinno and Schulz, 2019] focuses on the questions of why a certain atom is (not) in a given answer set and why no answer set can be computed. However, from an XAI perspective scenarios, where an ASP user finds themselves with a question about the behaviour of the ASP system, are more diverse than these plain settings.

In this survey, we provide an overview of the types of explanations that may arise in ASP and show how they are covered by existing ASP explanation approaches. Specifically:

– By taking an XAI perspective, we structure our survey along Local vs. Global explanations, distinguishing explanations for computed answer sets and explanations over the general behaviour of the given program.

– We identify a comprehensive list of typical user questions. While some of them can be natively addressed by existing

methods, some cannot. Thus, we show how they can be reduced to other questions where explanations are possible.

– Naturally, we cover approaches and tools that were not available for Fandinno and Schulz's survey.

– The user-centric view also allows us to highlight explanatory settings where current concepts and tools are lacking, even though those scenarios may be encountered by ASP users. We thus propose directions for future research to fill these gaps, as well as to address recent developments in AI.

This survey is intended to serve both researchers, and ASP users who are not experts, but are looking for explanation tools and methods which fit their situation and purpose.

2 Answer Set Programming

Syntax. A *(disjunctive) logic program* is a finite set of *rules*:

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \alpha_{k+1}, \ldots, \alpha_m, not\ \alpha_{m+1}, \ldots, not\ \alpha_n$$

where each α_i is an *atom* of form $p(t_1, \ldots, t_n)$, where p is a predicate, and each t_i is either a constant or a variable. A *literal* is either a formula α (*positive literal*) or $not\ \alpha$ (*negative literal*), where α is an atom. Intuitively, $not\ \alpha$ is true if α cannot be derived using rules, and false otherwise; not is called *default* negation. For a rule r of form $H(r) \leftarrow B(r)$, we refer to $H(r)$ as the *head* of r, and to $B(r)$ as the *body* of r. The *positive body atoms* of r are denoted by $B^+(r) = \{\alpha_{k+1}, \ldots, \alpha_m\}$, and the *negative body atoms* by $B^-(r) = \{\alpha_{m+1}, \ldots, \alpha_n\}$. A rule r is *normal* if $k = 1$ and a *constraint* if $k = 0$; Furthermore, r is *positive* if $n = m$ and a *fact* if $k = n = 1$ and it is variable-free. A rule is *ground* if all its literals are variable-free. A program P is *normal*, *positive* or *ground* if its rules are.Lastly, we use $not\ S = \{not\ s \mid s \in S\}$ for any set of atoms S as a shorthand.

Example 1. *Suppose the manager of the Bremen Town Musicians*[1] *relies on the following program P_1 about donkeys:*

$$sold(D) \leftarrow donkey(D), old(D), not\ musician(D) \quad (1)$$
$$musician(D) \leftarrow donkey(D), not\ sold(D) \quad (2)$$
$$old(D) \leftarrow donkey(D), age(D, A), A > 10 \quad (3)$$
$$donkey(d) \quad (4)$$
$$age(d, 25) \quad (5)$$

The program has three non-ground rules and two facts; intuitively, a donkey which is classified as old will be sold unless it is a musician, and a donkey is a musician unless it is sold. All rules are normal and P_1 is thus a normal program. However, one could also consider P_2 resulting from P_1 by replacing the first two rules with:

$$sold(D) \vee musician(D) \leftarrow donkey(D) \quad (6)$$
$$\leftarrow sold(D), not\ old(D) \quad (7)$$

Here (6) is a disjunctive rule and (7) a constraint. The choice of becoming a musician is now expressed via disjunction.

[1]Inspired by the Grimms' Fairy Tale

Semantics. The answer set semantics is defined via ground programs that are obtained by grounding as follows.

The *Herbrand universe* of a program P, denoted by HU_P, is the set of all constant symbols appearing in P, and the *Herbrand base* of P, denoted by HB_P, is the set of all ground atoms constructable with predicates from $\mathcal{P}$ and terms from HU_P. The *ground instances of a rule* $r \in P$, denoted by $grd(r, P)$, is the set of rules obtained by replacing all variables in r with constant symbols in HU_P in all possible ways. The *grounding* of P is then $grd(P) = \bigcup_{r \in P} grd(r, P)$.

An *interpretation* I of a ground program P is a subset of HB_P. Such I (i) *satisfies* a literal ℓ, denoted by $I \models \ell$, whenever $\alpha \in I$ if $\ell = \alpha$, and $\alpha \notin I$ if $\ell = not\ \alpha$, (ii) satisfies a rule $r \in P$ (denoted by $I \models r$) if $H(r) \cap I \neq \emptyset$ whenever $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$, and satisfies (is a *model* of) P (denoted by $I \models P$), if $I \models r$ for all $r \in P$. A model I of P is *minimal* if there is no $J \models P$ such that $J \subset I$.

The *Gelfond-Lifschitz* (GL-)*reduct* P^I of a program P relative to an interpretation I is the program obtained from $grd(P)$ when each rule $H(r) \leftarrow B^+(r), not\ B^-(r)$ (i) with $B^-(r) \cap I \neq \emptyset$ is deleted, and (ii) is replaced by $H(r) \leftarrow B^+(r)$, otherwise. Informally, the first step removes the rules where I contradicts some default negated literal, while the second step removes the negative body from all others.

An interpretation I is an *answer set* of a program P, if it is the minimal model of the GL-reduct P^I. By $AS(P)$ we denote the set of answer sets of a program P. A program P is *unsatisfiable* (or *inconsistent*) if $AS(P) = \emptyset$.

Example 2 (Ex. 1 cont'd)**.** *The Herbrand universe of P_1 is $HU_P = \{d, 25\}$. For the interpretation $I_1 = \{donkey(d), age(d, 25), old(d), sold(d)\}$, we have (among others) the following rules in the reduct $P_1^{I_1}$:*[2]

$$sold(d) \leftarrow donkey(d), old(d) \quad (8)$$
$$old(d) \leftarrow donkey(d), age(d, 25) \quad (9)$$
$$donkey(d) \quad (10)$$
$$age(d, 25) \quad (11)$$

I_1 is a minimal model of $P_1^{I_1}$ and thus $I_1 \in AS(P_1)$.

Language extensions. The *ASP-Core-2 Input Language Format* [Calimeri *et al.*, 2020] defines several language extensions; most are syntactic sugar but convenient in practice. We give here an informal account of some major constructs.

Aggregates are atoms of the form $\#aggr\ E \prec u$ where $\#aggr$ is the aggregate type ($\#count$, $\#sum$, $\#min$ or $\#max$), E is an aggregate set, $\prec$ is a comparison operator, and u is a term; they can appear only in rule bodies.

Example 3 (Ex. 1 cont'd)**.** *Let us add the facts $donkey(e)$, $age(e, 11)$ to P_1, yielding P_3. To express that at most one donkey can be a musician, we can add the constraint*

$$\leftarrow \#count\{D : musician(D)\} > 1 \quad (12)$$

The aggregate is true if the predicate $musician$ holds for more than one constant; then the constraint "fires" and prevents that the interpretation is an answer set.

[2]Rules which are trivially satisfied are not shown.

Choice rules are of the form $l \leq \{e_1; \ldots; e_n\} \leq u \leftarrow B$ and select between l and u atoms from $e_1, \ldots, e_n$; if $l = u$, we may write "$= u$".

Example 4 (Ex. 1 cont'd)**.** *The selection between $sold(D)$ and $musician(D)$ in (6) can be expressed with a choice rule:*

$$\{sold(D); musician(D)\} = 1 \leftarrow donkey(D) \tag{13}$$

The rule enforces that for each donkey D exactly one of $sold(D)$ and $musician(D)$ has to be true.

Weak constraints are of the form $:\sim B.\ [w, t_1, \ldots, t_m]$, where B is a set of literals, the t_i are constants or variables, and w is the *weight*. They incur a penalty w if B is satisfied, i.e., the constraint instance is violated. An answer set is then assigned the sum of all such penalties; it is *optimal*, if it has minimal penalty. More elaborated versions have priority levels, which allow to conveniently express model preference.

Example 5 (Ex. 3 cont'd)**.** *Instead of enforcing that only one of the two donkeys in P_3 can be a musician by the constraint (12), we can use also a weak constraint:*

$$:\sim musician(D)\ [1, D] \tag{14}$$

Each ground instance of (14) violated in an answer set incurs penalty 1. Hence, any optimal answer set will neither contain $musician(d)$ nor $musician(e)$ as both can be sold.

Further language extensions such as strong negation, conditional literals etc. ease problem encoding.

3 Forms of Explanations in ASP

Explainable AI (XAI) strives for making the behaviour of AI models, in particular of neural models such as classifiers, understandable to the user. While explainability and interpretability are often used interchangeably [Adadi and Berrada, 2018] and confused, the subject is usually either the output for a specific input, or insight into the model as a whole. This leads to *(local) explanations* (concerning a single data point) and *global explanations* (all data points), respectively, which in a logic-based setting can be defined using abduction principles [Marques-Silva, 2022], linking the notions to minimal satisfying assumptions and prime implicates.

We can align explainability of ASP with this distinction as follows. ASP programs are AI models, and answer sets of ASP programs are data points. Then *local explanations* refer to the case where the user has an answer set that they ask questions about. In contrast, *global explanations* address user questions about answer sets without having one at hand; here, the general program behaviour is of concern.

In the following, we will also speak of *justification*, which is sometimes used synonymously with the term *explanation*, and for our purposes we consider a justification a type of explanation.

3.1 Local Explanations

Local explanations in terms of ASP amount to explaining the reasoning behind an obtained answer set. In particular, given an answer set I of a program P, a user might ask questions about the presence or absence of certain atoms in I.

Q_1 : **Given $I \in AS(P)$, why $I \models \ell$ for some literal ℓ?**

These forms of questions are the most common explanation problems with several approaches addressing such questions. Here we will briefly summarize the concepts, categorizing each distinct approach as a specific method, denoted by **(M1)**, **(M2)**, etc. We start with those which were already covered by the previous survey of Fandinno and Schulz [2019]. For more details we refer to that work.

Justifications. *Offline justifications* **(M1)** [Pontelli *et al.*, 2009] are labelled directed graphs which, intuitively, can be seen as a step-by-step derivation of the truth value of the respective atom using the rules of the program, originally defined for ground, i.e., variable free, normal logic programs.

ABA-based justifications **(M2)** [Schulz and Toni, 2016] have their roots in *abstract argumentation* [Dung, 1995]. More specifically, they use arguments to describe the semantics of logic programs and the justifications are derived from trees of conflicting arguments (*attack trees*). A labeled assumption-based argumentation (ABA)-based answer set (LABAS) justification is a graph structure where the vertices are literals and edges are either attacks (representing conflicts between arguments) or supports (representing derivation relationships).

Causal justifications **(M3)** [Cabalar and Fandinno, 2017] provide explanations for literals in a given answer set based on a causal semantics for logic programs. The basis for causal justifications are algebraic terms representing joint causation through multiplication and alternative causes through addition. These causal terms can be presented as directed graphs showing the causal dependencies between literals.

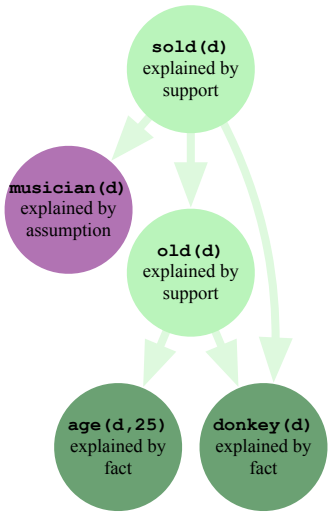
Justifications in rule-based ASP computation **(M4)** [Béatrix *et al.*, 2016] stem from the way the ASP solver ASPeRiX computes answer sets. Unlike to most solvers, it operates on the non-ground program and non-deterministically chooses rules rather than atoms. During this computation, the truth value of atoms is iteratively established and the *reason* for it being in the answer set is a set of ground rules which intuitively can be used to derive the atom in question.

Now we come to more recent methods or extensions.

Extended Justifications. The notion of offline justification has been recently extended to handle choice rules and constraints [Trieu *et al.*, 2021] and implemented in the xASP2 system, also supporting aggregates [Alviano *et al.*, 2024b].

One of the main changes in xASP2 is that it computes the *well-founded model* of the given program P first.[3] The explanation identifies which atoms are false in the well-founded model and determines a minimal set of additional atoms that must be assumed false. Together with the program facts, these allow deriving the atoms in I through rule application. The overall explanation is constructed in two phases: first, program rewriting and ASP meta encodings compute the minimal assumption set; second, these assumptions are used to generate a tree-like justification graph. The tree-like graph represents the rule application sequence yielding the target literal ℓ, where internal nodes correspond to rule applications and leaves are

[3] Briefly, the well-founded model $WFM(P)$ is a 3-valued approximation of $AS(P)$ where atoms true (resp. false) in $WFM(P)$ are true (resp. false) in each $I \in AS(P)$ [Van Gelder *et al.*, 1991].



(a) xASP2 explanation graph

```
*
|__sold(d)
|   |__donkey(d)
|   |__old(d)
|   |   |__donkey(d)
|   |   |__age(d,25)
```

(b) `xclingo` support graph

Figure 1: Explanation for $sold(d)$ in $I \in AS(P_1)$

either facts, atoms false in the well-founded model, or assumptions.

Example 6 (Ex. 1 cont'd)**.** *When we use xASP2 to explain why $sold(d)$ is in I, its browser displays the graph in Fig. 1a.*[4] *The (green) root node $sold(d)$ is connected via rule (1) to its positive body literals $donkey(d)$ and $old(d)$, which are also true in I and displayed in green. The negative body literal $musician(d)$ is false in I; this is represented as a purple assumption node, indicating it must be assumed false for the derivation to hold. The atom $old(d)$ (middle node) is further linked to its supporting facts $age(d, 25)$ and $donkey(d)$ dark green leaf nodes) via rule (3). This tree-like structure shows how $sold(d)$ is derived from the program facts together with the assumption that $musician(d)$ is false.*

Another novel approach is explanations via *unit-provable unsatisfiable subsets (1-PUS)* **(M5)**, which are the basis of the UCOREXPLAIN system [Alviano *et al.*, 2024a]. The concept of a 1-PUS is similar to the notion of minimal unsatisfiable sets [Alviano *et al.*, 2023] which is defined relative to a given set O of *objective atoms*, and is a minimal subset $U \subseteq O$ such that $P \cup \{\{o\} \geq 0 \leftarrow | \; o \in O\} \cup \{\leftarrow not\ u \mid u \in U\}$ has no answer sets. A 1-PUS differs by not checking the unsatisfiability of the amended program; instead, a simple set of (incomplete) derivation rules is used to check whether we can derive a contradiction. Furthermore, subset minimality is replaced with a user specified lexicographic order. In UCOREXPLAIN, the objective atoms O are auxiliary atoms that are injected into the rule bodies in the input program P. Intuitively, $o \in O$ controls whether a particular rule is enabled. Furthermore, the system adds a special rule to the program which makes it inconsistent whenever the literal ℓ we seek to explain is true. Hence, a 1-PUS in this setting represents a preferred set of rules which, in conjunction with simple derivation rules, can be used to derive ℓ.

[4]For space reasons, Fig. 1a shows a recreation.

The justifications provided by UCOREXPLAIN can then be displayed in a graph similar to xASP2. An important difference being that in xASP2, an atom a can only be justified by a rule r if $a \in H(r)$, which is not so in UCOREXPLAIN.

Explanation trees are obtained from a class of a vertex-labeled tree of a program, with variations such as shortest and k different explanations [Erdem and Oztok, 2015].

Support Graphs. *Support graphs* **(M6)** are defined over labelled programs, where each rule r is given a label $Lb(r)$ [Cabalar and Muñiz, 2024]. Furthermore, $LB(P) = \{Lb(r) \mid r \in P\}$ is the set of all labels in program P. For each atom p in an answer set I, they then define the notion of *support* which is defined as $SUP(P, I, p) = \{r \in P \mid p \in H(r), I \models B(r)\}$, i.e., the set of rules of P which have p in their head and have a true body w.r.t. I. Now, a support graph is a labelled directed graph $G = (I, E, \lambda)$ whose nodes are the atoms of the answer set I, edges $e \subseteq I \times I$ connect atoms, and $\lambda : I \to Lb(P)$ is an injective function assigning a label to each atom. Furthermore, it is required that for every $p \in I$, the rule r for which $\lambda(p) = Lb(r)$ satisfies $r \in SUP(P, I, p)$ and $B^+(r) = \{q \mid (q, p) \in E\}$. Intuitively, p is connected to all positive body atoms of r. If a support graph G is acyclic, it is called an *explanation*. The system `xclingo` uses support graphs to provide explanations [Cabalar and Muñiz, 2024]. It can handle choice rules, but due to the nature of support graphs, it can only give explanations for atom which are in the given answer set.

Example 7 (Ex. 1 cont'd)**.** *Figure 1b shows the output of `xclingo` to explain $sold(d) \in I$. The atom $sold(d)$ is connected to $donkey(d)$ and $old(d)$ to derive it using rule (1).*

Furthermore, `xclingo` allows the ASP programmer to attach natural language messages to rules via annotations (e.g., %# message "..."). These messages replace technical rule references in explanations, enabling domain-specific explanations for end-users without ASP expertise.

Witnesses. Similarly, *witnesses* **(M7)** [Wang *et al.*, 2022] are solver-independent but aim only at explaining positive literals. The notion aims at explaining why a set S of such literals is in a given answer set I of P by stepwise logical derivation, where in each step a set of rules in a reduct of P w.r.t. I augmented with already derived atoms is considered; for $S = I$, this amounts to explaining why I is an answer of P. For each step, a proof method, in particular resolution, may be chosen. In this way, an explanation of why $I = \{a, b\}$ is an answer set of $P = \{a \vee b,\ a \leftarrow b,\ b \leftarrow a\}$, can be given without resorting to guessing: for $S = \{a, b\}$, derive in step 1 a from $a \vee b.$ and $a \leftarrow b$ by resolution, and in step 2 b from a and $b \leftarrow a$. Recently, witnesses have been extended to programs with constraint atoms [Wang *et al.*, 2025].

Transform question into unsatisfiability. Interestingly, Q_1 can be transformed into the question of why a program has no answer set. For example, explaining why $a \in I$ reduces to explaining why $P \cup \{\leftarrow b \mid b \notin I\} \cup \{\leftarrow \neg b \mid b \in I, b \neq a\} \cup \{\leftarrow a\}$ has no answer sets. The intuition is that we add the literals true/false in I as assumptions except for a which is forced to be false. This idea was implicitly applied by Alviano *et al.* [2024a]. It should be noted that for programs with choice

rules the amended program may still be satisfiable. However, in such a case the literal is justified by the choice and there is no better explanation. Hence, explaining why a program is inconsistent, which is already of great interest by itself, could be a stepping stone to a more universal explanation approach.

Q_2 : **Given** $I \in AS(P)$**, why** $I \models \ell$ **but** $I \not\models \ell'$**?**
Miller [2019] argues that why-questions of form Q_1 are unequipped for providing proper explanations and instead questions should be seen as contrastive in the sense of "Why X and not Y?". In our setting this translates to asking Q_2. Although such a question can be transformed into unsatisfiability similar to above, which was considered for (weaker) questions of form "Why not Y?" [Bogatarkan and Erdem, 2020], dedicated explanation techniques exist in ASP.

Contrastive Explanations for ASP. Eiter *et al.* [2023] give a formal framework for *contrastive explanation* **(M8)** in ASP. In detail, their approach assumes that the user wants to know why for a set of atoms $E \subseteq I$ is included in the answer set instead of another set of atoms $F \cap I = \emptyset$. In order to provide an answer to this question, a *counterfactual account* is generated which consists of a program P' with answer set I' such that $E \cap I' = \emptyset$ and $F \subseteq I'$ and P' is "similar" to P. The latter means that P' retains as many rules as possible from P and the rules $r \in P' \setminus P$ are facts from a given set of assumables.

A counterfactual explanation is then a triple (Q_1, Q_2, Q_Δ), where Q_1 and Q_2 are essentially minimal sets of ground rules that derive E w.r.t. P and I, and F w.r.t. P' and I', respectively, and $Q_\Delta = P \setminus P'$ is the set of rules that were removed.

The contrastive explanation results from the counterfactual explanation by taking the symmetric difference of Q_1 and Q_2 and removing the (optional) fixed rules, which must be in P'.

Example 8 (Ex. 1 cont'd)**.** *Consider again P_1 and assume we add the following constraint yielding P_4:*

$$\leftarrow musician(D), age(D, A), A > 21 \tag{15}$$

Now, the grounding of the program P_4 includes the following rules:

$$sold(d) \leftarrow donkey(d), old(d), not\ musician(d) \tag{16}$$
$$musician(d) \leftarrow donkey(d), not\ sold(d) \tag{17}$$
$$old(d) \leftarrow donkey(d), age(d, 25), 25 > 10 \tag{18}$$
$$\leftarrow musician(d), age(d, 25), 25 > 21 \tag{19}$$

It holds that $AS(P_4) = \{I_1\}$. Suppose a user asks why $sold(d) \in I_1$ rather than $musician(d) \in I_1$, and the facts are fixed. The only contrastive explanation is then $(\{(16), (18)\}, \{(17)\}, \{19)$ intuitively meaning that (16) and (18) explain why $sold(d) \in I_1$, but if (19) were removed, we could get $musician(d) \in I_1$ via (17).

Q_3 : **Why is answer set I of P reported as optimal?**
For programs with weak constraints, explanations for optimality would be of great interest. Although currently no method is specifically tailored to this question, it is possible to reduce it to checking why there is no answer set that is better. This can be done by introducing a constraint enforcing strictly less penalty than I. Explanation methods for the non-existence of answer sets, which we discuss later, can then be employed. While we classified this question as local, it could also be seen as global as the only thing we take from I is its cost.

3.2 Global Explanations

Global explanations aim to explain the model and its logic as a whole [Adadi and Berrada, 2018]. In terms of ASP, this concerns the following questions.

Q_4 : **Why is a (not) in some answer set of P?**
Explaining why a appears, or does not appear, in some answer set of P can be seen as local where the I is left implicit. Hence, a local explanation, of a computed witness I, can be given. However, obtaining global explanations without needing a specific answer set at hand is also possible.

Why-not Provenance. This question has been tackled by *why-not provenance* **(M9)** [Damásio *et al.*, 2013] which provides justifications for the truth values of atoms w.r.t. answer set semantics for normal logic programs. This notion has formal similarities with causal justifications, but rather than causal reasons it gives possible modifications of the program that are needed to ensure a certain truth value.

Top-down Computation (M10) The Constraint Answer Set Solving system *s(CASP)* [Arias *et al.*, 2018] can provide justification trees for atoms contained in some answer set due to its implementation in Prolog. The system finds stable models by goal-directed search and top-down evaluation operating on the non-ground program. This means that the user essentially queries for a certain – potentially negated – predicate and is given an answer set where a particular ground instance of the predicate is contained. The process is top-down in the sense that it starts from the queried predicate and looks whether a ground instance can be justified in an answer set. This has the side effect of producing a justification tree which can be displayed to the user. Furthermore, this explanation also operates on the non-ground program, thus potentially yielding more compact explanations.

Example 9 (Ex. 1 cont'd)**.** *If we run s(CASP) for the ground query $sold(d)$ on P_1 and ask for a justification, it outputs*

```
'sold' holds (for d), because
 'donkey' holds (for d), and
 'old' holds (for d), because
  'donkey' holds (for d), justified above,
  and 'age' holds (for d, and 25).
 there is no evidence that 'musician' holds
 (for d), because
  'donkey' holds (for d), justified above,
  and it is assumed that 'sold' holds (for d)
```

In a similar fashion, the Satoh-Iwayama procedure [Satoh and Iwayama, 1992; Dauphin and Satoh, 2019], which is also goal-directed, can be used to answer Q_4. It also produces a derivation tree that can be given as a justification, but different from s(CASP), the procedure operates on ground programs.

Q_5 : **Why is a in no or in every answer set of P?**
Dedicated explanation concepts related to questions of this form are *proof systems* **(M11)** for ASP. For example, Bonatti [2001] introduced a resolution calculus for sceptical

entailment, which is when a appears in every answer set, and a tableaux calculus was introduced by Gebser and Schaub [2013]. Recently, Eiter and Geibinger [2025] presented a sequent calculus for sceptical entailment in equilibrium logic, which is a logical underpinning of ASP.

Similar to the questions in the previous section, answering why an atom a is in no (respectively, every) answer set of P can be reduced to explaining why $P \cup \{\leftarrow \neg a\}$ (respectively, $P \cup \{\leftarrow a\}$) has no answer set.

Q_6 : **Why does P have no answer sets?**
Generally, the works studying this question concern *debugging*, i.e., they aim not at a user seeking general explanations but rather to provide technical feedback to the ASP engineer who was expecting a different result, or *proof logging*, where solvers provide verifiable certificates for their results.

Debugging. One avenue of research into debugging is given by *meta-encoding techniques* **(M12)** which, intuitively, provide reasons why each interpretation is not an answer set. Systems that implement this approach are spock [Brain *et al.*, 2007; Gebser *et al.*, 2008] and Ouroboros [Oetsch *et al.*, 2010]; for the latter no a-priori knowledge of inconsistency is needed when the user provides an interpretation whereas spock needs no such information. Another approach is to remove the inconsistency by utilizing *minimal unsatisfiable sets (MUS)* **(M13)**, as done e.g. in DWASP [Dodaro *et al.*, 2019]. A minimal unsatisfiable set contains (non-ground) rules of the program which cannot jointly hold, but if one of them were removed, then the program would be consistent. This concept has roots in model-based diagnosis and constraint satisfaction and has been generalized to nonmonotonic logics including ASP [Eiter *et al.*, 2014; Brewka *et al.*, 2019] taking the nonmonotonicity of the stable model semantics into account. *Stepping* **(M14)** [Oetsch *et al.*, 2018] is an interactive debugging approach that lets the user build an intended answer set step by step, illustrating where the process fails. For more details on debugging techniques we refer to the survey of Fandinno and Schulz [2019].

Proof Systems. For Q_5, we mentioned proof systems for ASP, which can also serve as a basis to answer question Q_6, as it is essentially asking why $\bot$ is derivable.

Furthermore, for certifying the result whenever a solver reports "unsat", specialised *proof logging* systems for SAT were adapted to ASP [Alviano *et al.*, 2019; Chew *et al.*, 2024]. Those systems output resolution proofs for the SAT clauses generated by the solver to derive $\bot$.

Abstraction. In contrast to debugging and proof approaches, *abstraction* **(M15)** notions in ASP highlight the reason for unsolvability through omitting or clustering irrelevant details [Saribatur and Eiter, 2021; Saribatur *et al.*, 2021]. Here, abstraction is considered as an *over-approximation*, where a program P' over $\mathcal{A}'$ is called an *abstraction* of a program P over $\mathcal{A}$, if there exists a mapping $m : \mathcal{A} \to \mathcal{A}' \cup \{\top\}$ such that for each answer set I of P, $I' = \{m(l) \mid l \in I\}$ is an answer set of P', i.e., $m(AS(P)) \subseteq AS(P')$. *Omission-based abstraction* [Saribatur and Eiter, 2021] relies on a mapping $m_A : \mathcal{A} \to \mathcal{A} \cup \{\top\}$ for a set A of atoms to be omitted, such that $m_A(\alpha) = \top$ if $\alpha \in A$ and $m_A(\alpha) = \alpha$ if $\alpha \in \mathcal{A} \setminus A$. Thus $m_A(AS(P))$ amounts to projecting away the omitted atoms, denoted as $AS(P)|_{\overline{A}}$. An abstract program $omit(P, A)$ is constructed by a syntactic operator that removes rules with heads in A and add choice rules to the head of rules with body atoms occurring in A. This ensures over-approximation of the (projected) answer sets. An *(answer set) blocker set* refers to a set R of atoms that are kept in the vocabulary, while the rest is omitted from a ground program P to obtain the program $omit(P, \mathcal{A} \setminus R)$, such that $omit(P, \mathcal{A} \setminus R)$ is also unsatisfiable; the atoms in R are thus blocking occurrence of answer sets.

Example 10 (Ex. 8 cont'd)**.** *Assume we add the following rules to P_4, yielding P_5 that is unsatisfiable:*

$$\leftarrow not\ musician(D), canSing(D) \quad (20)$$
$$canSing(d) \quad (21)$$

A blocker set for the grounding of the program P_5 would be $R = \{donkey(d), musician(d), age(d, 25), canSing(d)\}$, with resulting abstraction $omit(grd(P_5), \mathcal{A} \setminus R)$ as below that is still unsatisfiable.

$$\{musician(d)\} \leftarrow donkey(d) \quad (22)$$
$$\leftarrow musician(d), age(d, 25), 25 > 21 \quad (23)$$
$$\leftarrow not\ musician(d), canSing(d) \quad (24)$$
$$canSing(d) \quad (25)$$
$$donkey(d) \quad (26)$$
$$age(d, 25) \quad (27)$$

In fact R would be the minimal blocker set, as omitting further atoms would no longer preserve unsatisfiability.

A CEGAR-style [Clarke *et al.*, 2003] abstraction and refinement methodology enables automated computation of these atom sets. This is lifted to the non-ground case for clustering of domain elements [Saribatur *et al.*, 2021] to find abstractions that "zoom-in" on the unsatisfiable part of the program.

3.3 Summary

Table 1 presents an overview of the explanation techniques we addressed in this section; the symbol * means one can reduce the question to another one and use the latter's method, like how we can explain why an atom is in some answer set (Q_4) by providing a witness and using it for the explanation (Q_1). We show which questions these techniques can explain, which program type they support, whether they support explaining positive or negative atoms (or both), if they are solver-dependent, and their explanation output. We can observe that all of the questions can be explained natively or via transforming to another question. While Q_3 is the only question with no native explanation, Q_2 and Q_5 have one dedicated technique, and the rest of the questions have multiple supporting techniques. Observe that many of the questions can be reduced to explaining unsatisfiability. Most of the methods can explain both truth values, and there is a balanced view of solver-dependency. We see quite a diversity in the explanation outputs.

Other Related Explanation Methods
The system `fasp` [Fichte *et al.*, 2022] allows for navigation toward desired partial solutions (called *facets*) in ASP, which

Method (System)	Type	Questions	Prog.	Pos./Neg.	Dep.	Output
(M1) Offline Justifications (xASP)	L	Q_1, Q_4^*	Normal	Both	No	Graph
(M2) ABA-based Justifications (LABAS)	L	Q_1, Q_4^*	Normal	Both	No	Argumentation graph
(M3) Causal Justifications	L	Q_1, Q_4^*	Normal	Both	No	Graph / algebraic term
(M4) Rule-based Justifications	L	Q_1, Q_4^*	Normal	Both	Yes	Set of rules
(M5) 1-PUS (UCOREXPLAIN)	L	Q_1, Q_4^*	Disj.	Both	No	Graph of derivations
(M6) Support Graphs (xclingo)	L	Q_1, Q_4^*	Disj.	Positive	Yes	Graph
(M7) Witnesses	L	Q_1, Q_4^*	Disj.	Positive	No	Sequence of rule sets
(M8) Contrastive Explanations	L	Q_2	Disj.	Both	No	Structured explanation
(M9) Why-not Provenance	G	Q_4	Normal	Both	No	Provenance graph
(M10) Top-down Computation (s(CASP))	G	Q_1^*, Q_4	Normal	Both	Yes	Justification tree
(M11) Proof Systems	G	$Q_1^*, Q_2^*, Q_3^*, Q_5, Q_6$	Disj.	n/a	Yes/No	Proof certificate
(M12) Debugging (spock, Ouroboros)	G	$Q_1^*, Q_2^*, Q_3^*, Q_5^*, Q_6$	Disj.	n/a	No	Meta-interpretation
(M13) MUS-based Debugging (DWASP)	G	$Q_1^*, Q_2^*, Q_3^*, Q_5^*, Q_6$	Disj.	n/a	No	Min. unsat. rule sets
(M14) Stepping	G	$Q_1^*, Q_2^*, Q_3^*, Q_5^*, Q_6$	Disj.	n/a	No	Interactive trace
(M15) Abstraction	G	$Q_1^*, Q_2^*, Q_3^*, Q_5^*, Q_6$	Normal	n/a	No	Abstracted program

Table 1: Overview of explanation techniques in Answer Set Programming.

recently was extended to navigate the solution space of ASP-encoded planning tasks [Gnad *et al.*, 2025]. Facets are atoms that occur in some but not all answer sets. Interactively enforcing or forbidding such facets to see which properties hold allows the user to better understand the solution space. A further recent answer set explorer is *viasp* [Glaser *et al.*, 2025].

Model reconciliation is a form of contrastive explanation by reconciling the system's and the user's model (i.e., programs) to agree on a property of the made decision, which was also considered in ASP with a focus on planning [Son *et al.*, 2021; Nguyen *et al.*, 2020].

Tool	Disjunction	Choice rules	Weak const.	Aggr.
LABAS	no	no	no	no
s(CASP)	no	no	no	no
UCOREX.	no	yes	no	no
xclingo	yes	yes	no	no
xASP2	no	yes	no	yes
spock	no	no	no	no
Ouroboros	yes	yes	no	no
DWASP	yes	yes	yes	yes

Table 2: Language support of existing tools in Explainable ASP

4 Outlook for Explainable ASP

Having reviewed the existing ASP explanation approaches, we spot gaps and present issues for future research.

4.1 Practical Aspects

Notably, there are more explanation concepts than actual software tools implementing them. Issues with the tools include the following ones.

Language Features. As shown in Section 2, disjunction and language extensions can help with representing challenging real-world problems. Table 2 summarizes the language support of available explanation tools. Although, with recent advances, there are more tools supporting language extensions, there are still some gaps. Disjunction is often not supported by the tools, because the underlying explanation method cannot generally handle it. The reason for that being the increase in expressiveness when one adds disjunctive rules with head-cycles. Support for aggregates and choice rules has increased, but most tools still do not support them or have some syntactic restriction on them. Weak constraints are still generally not supported. Here DWASP is the exception, though this is a debugging tool - supporting weak constraints is independent from a diagnosis on how to restore consistency.

System Divide. The methods implemented in current systems either focus on providing local explanations or global ones. This puts the burden of deciding on which system(s) to employ on the user, depending on the questions they have and may cause undesired overhead.

Scalability/Non-ground ASP. Since most of these approaches can be considered as *post-hoc*, meaning that they are for explaining an encountered outcome, e.g., an answer set or none, the techniques need to reconstruct the reasoning that achieves the result in order to explain it. This of course causes issues especially if the problem at hand is large or difficult to solve. Given that providing explanations has high computational complexity in general [Eiter *et al.*, 2023; Wang *et al.*, 2022], it is important to find ways to tackle scalability, e.g., by incorporating explanation techniques into the ASP solvers. Also most explanation approaches operate on ground programs where all variables have been instantiated, except for s(CASP), which also justifies the instantiation of the variables. While ASP solvers also often only operate on ground programs, in the scope of explanation, preserving the relationship between ground atoms that were obtained from instantiating the same predicate may be beneficial.

Complex User Questions. Currently, in case of questions that arise over a collection of answer sets, the only way to address them is to get explanations for each answer set. However, when a user asks Q_7*: Why is* a *(not) in answer sets* $I_1, \ldots, I_n$*?*, they expect a global explanation on the behaviour of the program that obtains a in those particular answer sets, which currently cannot be obtained. Explanations with respect to the quantity of answer sets, which may be needed for in-

dustrial settings with configuration problems etc., also cannot be addressed through any of the approaches. These questions can be of form Q_8 : *Why does P have so many/few answer sets?* and likely need dedicated solutions. Facets [Fichte *et al.*, 2022] could be a worthwhile technique when constructing such explanations. Observe that these questions can also be seen as contrastive, as the user has encountered a number of answer sets against their expectation. Thus explanations on under-/over-restrictions would be of use. Furthermore, as *symmetry* is another cause for many answers, explanations that also show the detected symmetries might be helpful.

4.2 Cognitive Aspects

Here, we consider cognitive aspects of explanations in ASP.

Conciseness. One challenge is reducing information complexity of explanations for large programs, as they may contain many rules to trace. Providing more concise explanations from a possibly higher-level view remains open. Abstraction can be helpful to achieve this, as considered in other domains, e.g., planning [Sreedharan *et al.*, 2021] and mathematical reasoning [Poesia Reis e Silva and Goodman, 2022]. Recent work reports a dual effect of applying abstraction in ASP explanations: abstraction by clustering helps in understanding, while omission reduces cognitive effort [Saribatur *et al.*, 2026]. These results support the need for further investigations on obtaining concise explanations.

Interpretability. An important aspect when providing explanations is whether the provided explanation is indeed understandable to the user, especially if they are not experts. While justifications or similar techniques aim to provide the reasoning behind the made decision, contrastive explanations have insight into the epistemic state of the explainee and can focus on the actual core of the misunderstanding [Miller, 2019].

Regarding explaining inconsistency, due its difficult nature, finding a satisfying answer is challenging. Concepts from debugging, like minimal unsatisfiable subsets or proof logging may be used, but it is unclear how respective rule sets should be presented to a non-expert. Notably, the idea was successfully applied for ASP-based product configuration [Herud *et al.*, 2022], which is a domain where minimal unsatisfiable subsets can easily be translated into an explanation. Step-wise explanation notions [Bogaerts *et al.*, 2020; Oetsch *et al.*, 2018] could be beneficial for understanding how unsatisfiability in ASP is brought about.

Taking into account the aim of conciseness, one idea for explaining inconsistency is by some hidden "lemma". For example, a graph colouring instance may be inconsistent due to some clique in the graph. Revealing this clique would be an adequate explanation for users with sufficient domain knowledge. However, how such explanation could be derived is unknown. Abstraction could be useful but also investigating interpolation, which could produce the lemma, might be worthwhile [Gabbay *et al.*, 2010].

Interaction and Communication. Another issue with the available explanation techniques is the lack of user-friendly tools that allow one to obtain explanations without much technical expertise. In order to communicate the highly technical explanation to the end-user, Large Language Models (LLMs) might be of help. This has been explored in preliminary work on explanations for guess-and-check programs where questions are parsed by an LLM and the resulting MUS is translated by the LLM into natural language [Geibinger *et al.*, 2024].

Recently LLMs and symbolic solvers were bridged through the Model Context Protocol (MCP) [Szeider, 2025]. It enables LLMs to access formal reasoning capabilities including ASP, showing potential in achieving a user-friendly interaction for ASP explanations. Furthermore, it remains to be explored whether LLMs can be used to find explanations for ASP.

5 Conclusion

We have provided an overview of explanations in ASP from an XAI perspective, structured along explanation types and on what questions the ASP user might find themselves with. We profiled a number of existing explanation methods and existing systems that can be used off-the-shelf to address the user questions. Overall, there is a rich landscape of methods and systems available with diversity in explanation types, question support, and explanation formats, but none fits all needs.

We further presented issues from practical and cognitive perspectives to address, some of which concern limitation in language support, complex user questions, and, importantly, interpretability and communication. Notably, most of the user questions can be transformed into unsatisfiability; thus advanced explanations for program inconsistency could provide a stepping stone to more universal explanation approaches. That and connecting to latest AI research poses a challenge.

Acknowledgments

This research was funded in whole or in part by the Austrian Science Fund (FWF) grants 10.55776/COE12 and 10.55776/T1315, the Vienna Science and Technology Fund (WWTF) grant 10.47379/ICT25044, and benefited from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101034440 (LogiCS@TUWien).

Tobias Geibinger is a recipient of a DOC Fellowship of the Austrian Academy of Sciences.

References

[Adadi and Berrada, 2018] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). *IEEE access*, 6:52138–52160, 2018.

[Alviano *et al.*, 2019] Mario Alviano, Carmine Dodaro, Johannes Klaus Fichte, Markus Hecher, Tobias Philipp, and Jakob Rath. Inconsistency proofs for ASP: the ASP - DRUPE format. *TPLP*, 19(5-6):891–907, 2019.

[Alviano *et al.*, 2023] Mario Alviano, Carmine Dodaro, Salvatore Fiorentino, Alessandro Previti, and Francesco Ricca. ASP and subset minimality: Enumeration, cautious reasoning and muses. *AIJ*, 320:103931, 2023.

[Alviano *et al.*, 2024a] Mario Alviano, Susana Hahn, Orkunt Sabuncu, and Hannes Weichelt. Answer set explanations via preferred unit-provable unsatisfiable subsets. In *Proc. LPNMR*, pages 187–199, 2024.

[Alviano *et al.*, 2024b] Mario Alviano, Ly Ly T. Trieu, Tran Cao Son, and Marcello Balduccini. The XAI system for answer set programming xASP2. *J. Log. Comput.*, 34(8):1500–1525, 2024.

[Arias *et al.*, 2018] Joaquín Arias, Manuel Carro, Elmer Salazar, Kyle Marple, and Gopal Gupta. Constraint answer set programming without grounding. *TPLP*, 18(3-4):337–354, 2018.

[Béatrix *et al.*, 2016] Christopher Béatrix, Claire Lefèvre, Laurent Garcia, and Igor Stéphan. Justifications and Blocking Sets in a Rule-Based Answer Set Computation. In *Proc. ICLP*, pages 6:1–6:15, 2016.

[Bogaerts *et al.*, 2020] Bart Bogaerts, Emilio Gamba, Jens Claes, and Tias Guns. Step-wise explanations of constraint satisfaction problems. In *Proc. ECAI*, pages 640–647, 2020.

[Bogatarkan and Erdem, 2020] Aysu Bogatarkan and Esra Erdem. Explanation generation for multi-modal multi-agent path finding with optimal resource utilization using answer set programming. *TPLP*, 20(6):974–989, 2020.

[Bonatti, 2001] Piero A. Bonatti. Resolution for skeptical stable model semantics. *J. Autom. Reason.*, 27(4):391–421, 2001.

[Brain *et al.*, 2007] Martin Brain, Martin Gebser, Jörg Pührer, Torsten Schaub, Hans Tompits, and Stefan Woltran. Debugging ASP programs by means of ASP. In *Proc. LPNMR*, pages 31–43. Springer, 2007.

[Brewka *et al.*, 2011] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[Brewka *et al.*, 2019] Gerhard Brewka, Matthias Thimm, and Markus Ulbricht. Strong inconsistency. *AIJ*, 267:78–117, 2019.

[Cabalar and Fandinno, 2017] Pedro Cabalar and Jorge Fandinno. Enablers and inhibitors in causal justifications of logic programs. *TPLP*, 17(1):49–74, 2017.

[Cabalar and Muñiz, 2024] Pedro Cabalar and Brais Muñiz. Model explanation via support graphs. *TPLP*, 24(6):1109–1122, 2024.

[Calegari *et al.*, 2020] Roberta Calegari, Giovanni Ciatto, and Andrea Omicini. On the integration of symbolic and sub-symbolic techniques for XAI: A survey. *Intelligenza Artificiale*, 14(1):7–32, 2020.

[Calimeri *et al.*, 2020] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca, and Torsten Schaub. ASP-Core-2 input language format. *TPLP*, 20(2):294–309, 2020.

[Chew *et al.*, 2024] Leroy Chew, Alexis de Colnet, and Stefan Szeider. ASP-QRAT: A conditionally optimal dual proof system for ASP. In *Proc. KR*, 2024.

[Clarke *et al.*, 2003] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

[Damásio *et al.*, 2013] Carlos Viegas Damásio, Anastasia Analyti, and Grigoris Antoniou. Justifications for logic programming. In *Proc. LPNMR*, pages 530–542, 2013.

[Dauphin and Satoh, 2019] Jérémie Dauphin and Ken Satoh. Explainable ASP. In *Proc. PRIMA*, pages 610–617, 2019.

[Dodaro *et al.*, 2019] Carmine Dodaro, Philip Gasteiger, Kristian Reale, Francesco Ricca, and Konstantin Schekotihin. Debugging non-ground ASP programs: Technique and graphical tools. *TPLP*, 19(2):290–316, 2019.

[Dung, 1995] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *AIJ*, 77(2):321–358, 1995.

[Eiter and Geibinger, 2025] Thomas Eiter and Tobias Geibinger. A sequent calculus for answer set entailment. In *Proc. IJCAI*, pages 4463–4472. ijcai.org, 2025.

[Eiter *et al.*, 2014] Thomas Eiter, Michael Fink, Peter Schüller, and Antonius Weinzierl. Finding explanations of inconsistency in multi-context systems. *AIJ*, 216:233–274, 2014.

[Eiter *et al.*, 2023] Thomas Eiter, Tobias Geibinger, and Johannes Oetsch. Contrastive explanations for answer-set programs. In *Proc. JELIA*, pages 73–89, 2023.

[Erdem and Oztok, 2015] Esra Erdem and Umut Oztok. Generating explanations for biomedical queries. *TPLP*, 15(1):35–78, 2015.

[Fandinno and Schulz, 2019] Jorge Fandinno and Claudia Schulz. Answering the "why" in answer set programming - A survey of explanation approaches. *TPLP*, 19(2):114–203, 2019.

[Fichte *et al.*, 2022] Johannes Klaus Fichte, Sarah Alice Gaggl, and Dominik Rusovac. Rushing and strolling among answer sets–navigation made easy. In *Proc. AAAI*, pages 5651–5659, 2022.

[Gabbay *et al.*, 2010] Dov M. Gabbay, David Pearce, and Agustín Valverde. Interpolation in equilibrium logic and answer set programming: the propositional case. *CoRR*, abs/1012.3947, 2010.

[Gebser and Schaub, 2013] Martin Gebser and Torsten Schaub. Tableau calculi for logic programs under answer set semantics. *ACM TOCL*, 14(2):15:1–15:40, 2013.

[Gebser *et al.*, 2008] Martin Gebser, Jörg Pührer, Torsten Schaub, and Hans Tompits. A meta-programming technique for debugging answer-set programs. In *Proc. AAAI*, volume 8, pages 448–453, 2008.

[Geibinger *et al.*, 2024] Tobias Geibinger, Johannes Oetsch, and Tobias Kaminski. Explanations for guess-and-check ASP encodings using an LLM (extended abstract). In *Proc. TAASP*, 2024.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *NGC*, 9(3):365–385, 1991.

[Glaser *et al.*, 2025] Luis Glaser, Javier Romero, Torsten Schaub, and Stephan Zwicknagl. viasp: A visualization

tool for answer set programming (extended abstract). In *Proc. TAASP*, 2025.

[Gnad *et al.*, 2025] Daniel Gnad, Markus Hecher, Sarah Alice Gaggl, Dominik Rusovac, David Speck, and Johannes K Fichte. Interactive exploration of plan spaces. In *Proc. KR*, pages 599–609, 2025.

[Herud *et al.*, 2022] Konstantin Herud, Joachim Baumeister, Orkunt Sabuncu, and Torsten Schaub. Conflict handling in product configuration using answer set programming. In *Proc. ASPOCP@ICLP*, volume 3193 of *CEUR*, 2022.

[Kareem *et al.*, 2024] Irfan Kareem, Katie Gallagher, Manuel Borroto, Francesco Ricca, and Alessandra Russo. Using learning from answer sets for robust question answering with LLM. In *Proc. LPNMR*, pages 112–125, 2024.

[Lifschitz, 2019] Vladimir Lifschitz. *Answer Set Programming*. Springer, 2019.

[Marques-Silva, 2022] João Marques-Silva. Logic-based explainability in machine learning. In *Proc. RW*, pages 24–104, 2022.

[Miller, 2019] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *AIJ*, 267:1–38, 2019.

[Nguyen *et al.*, 2020] Van Nguyen, Vasileiou Loukas Stylianos, Tran Cao Son, and William Yeoh. Explainable planning using answer set programming. In *Proc. KR*, pages 662–666, 2020.

[Oetsch *et al.*, 2010] Johannes Oetsch, Jörg Pührer, and Hans Tompits. Catching the ouroboros: On debugging non-ground answer-set programs. *TPLP*, 10(4-6):513–529, 2010.

[Oetsch *et al.*, 2018] Johannes Oetsch, Jörg Pührer, and Hans Tompits. Stepwise debugging of answer-set programs. *TPLP*, 18(1):30–80, 2018.

[Poesia Reis e Silva and Goodman, 2022] Gabriel Poesia Reis e Silva and Noah Goodman. Left to the reader: Abstracting solutions in mathematical reasoning. In *Proc. CogSci*, 2022.

[Pontelli *et al.*, 2009] Enrico Pontelli, Tran Cao Son, and Omar Elkhatib. Justifications for logic programs under answer set semantics. *TPLP*, 9(1):1–56, 2009.

[Rader and Russo, 2025] Alexander Rader and Alessandra Russo. A survey of neurosymbolic answer set programming. *NAI*, 2025.

[Rajaratnam *et al.*, 2023] David Rajaratnam, Torsten Schaub, Philipp Wanko, Kai Chen, Sirui Liu, and Tran Cao Son. Solving an industrial-scale warehouse delivery problem with answer set programming modulo difference constraints. *Algorithms*, 16(4):216, 2023.

[Rajasekharan *et al.*, 2023] Abhiramon Rajasekharan, Yankai Zeng, Parth Padalkar, and Gopal Gupta. Reliable natural language understanding with large language models and answer set programming. In *Proc. ICLP*, pages 274–287, 2023.

[Saribatur and Eiter, 2021] Zeynep G. Saribatur and Thomas Eiter. Omission-based abstraction for answer set programs. *TPLP*, 21(2):145–195, 2021.

[Saribatur *et al.*, 2021] Zeynep G. Saribatur, Thomas Eiter, and Peter Schüller. Abstraction for non-ground answer set programs. *AIJ*, 300:103563, 2021.

[Saribatur *et al.*, 2026] Zeynep G. Saribatur, Johannes Langer, and Ute Schmid. The dual role of abstracting over the irrelevant in symbolic explanations: Cognitive effort vs. understanding. In *Proc. CogSci*, 2026.

[Satoh and Iwayama, 1992] Ken Satoh and Noboru Iwayama. A correct goal-directed proof procedure for a general logic program with integrity constraints. In *Proc. ELP*, volume 660 of *LNCS*, pages 24–44, 1992.

[Schaub and Woltran, 2018] Torsten Schaub and Stefan Woltran. Special issue on answer set programming. *KI*, 32:101–103, 2018.

[Schulz and Toni, 2016] Claudia Schulz and Francesca Toni. Justifying answer sets using argumentation. *TPLP*, 16(1):59–110, 2016.

[Schwalbe and Finzel, 2024] Gesina Schwalbe and Bettina Finzel. A comprehensive taxonomy for explainable artificial intelligence: a systematic survey of surveys on methods and concepts. *DMKD*, 38(5):3043–3101, 2024.

[Shakarian *et al.*, 2023] Paulo Shakarian, Chitta Baral, Gerardo I Simari, Bowen Xi, and Lahari Pokala. *Neuro Symbolic Reasoning and Learning*. Springer, 2023.

[Son *et al.*, 2021] Tran Cao Son, Van Nguyen, Stylianos Loukas Vasileiou, and William Yeoh. Model reconciliation in logic programs. In *Proc. JELIA*, pages 393–406, 2021.

[Sreedharan *et al.*, 2021] Sarath Sreedharan, Siddharth Srivastava, and Subbarao Kambhampati. Using state abstractions to compute personalized contrastive explanations for AI agent behavior. *AIJ*, 301:103570, 2021.

[Szeider, 2025] Stefan Szeider. Bridging language models and symbolic solvers via the model context protocol. In *Proc. SAT*, pages 30–1, 2025.

[Trieu *et al.*, 2021] Ly Ly T. Trieu, Tran Cao Son, and Marcello Balduccini. exp(ASPc) : Explaining ASP programs with choice atoms and constraint rules. In *Proc. ICLP*, pages 155–161, 2021.

[Van Gelder *et al.*, 1991] Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3):619–649, 1991.

[Wang *et al.*, 2022] Yisong Wang, Thomas Eiter, Yuanlin Zhang, and Fangzhen Lin. Witnesses for answer sets of logic programs. *ACM TOCL*, 2022.

[Wang *et al.*, 2025] Yisong Wang, Xianglong Wang, Zhongtao Xie, and Thomas Eiter. Witnesses for answer sets of basic logic programs. In *Proc. IJCAI*, pages 4696–4705, 2025.