Explanations for Answer Set Programming

Mario Alviano
DEMACS, University of Calabria,
Via Bucci 30/B, 87036 Rende (CS), Italy
mario.alviano@unical.it

Ly Ly Trieu & Tran Cao Son
New Mexico State Universty
NM, USA
lytrieu|stran@nmsu.edu

Marcello Balduccini
Saint Joseph's University
PA , USA
mbalducc@sju.edu

The paper presents an enhancement of `xASP`, a system that generates explanation graphs for Answer Set Programming (ASP). Different from `xASP`, the new system, `xASP2` , supports different `clingo` constructs like the choice rules, the constraints, and the aggregates such as *#sum*, *#min*. This work formalizes and presents an explainable artificial intelligence system for a broad fragment of ASP, capable of shrinking as much as possible the set of assumptions and presenting explanations in terms of directed acyclic graphs.

1 Introduction

Recently, many modern artificial intelligence systems are increasingly capable of tackling complex problems. However, their lack of transparency can create a new issue: users may not comprehend why a solution was obtained, making it difficult to trust the results. Moreover, with the *right to an explanation* law extensively discussed in the USA, EU, and UK, and partly enacted in some countries, explainable artificial intelligence (XAI) has experienced a substantial increase in interest. Thus, the focus of this paper is on the development of an XAI system for Answer Set Programming (ASP) [10, 13]. Answer Set Programming (ASP) [10, 13] is a well-known paradigm for problem-solving using logic programs under answer set semantics [7] in knowledge representation and reasoning (KR&R) and an extension of Datalog with a strong connection with well-founded semantics [14]. A variety of applications such as planning, diagnosis, etc, have been successfully implemented using it. In this paper, our goal is to provide an answer to the question "*given an answer set A of a program* Π *and an atom* α*, why an atom* α *is true (or false) in A?*".

The emergence of XAI has brought significant attention from researchers in ASP community, resulting in numerous proposed systems aimed at addressing this issue such as xclingo [4], `DiscASP` [9], `xASP` [19], `exp(ASP`$^{\mathtt{c}}$`)` [18]. However, these systems are incapable of handling one or more of the following scenarios: (*i*) false atoms can be explained by the system, (*ii*) the ability to support certain advanced language features. In this paper, we proposed an improvement system, called `xASP2` , that takes inspiration from the approach used in `xASP` [19] and [16]. `xASP2` are able to substantially increase scalability and breadth of supported language features while producing explanation graphs with more immediately and consistently useful to users. The improvement presented here deals with two main issues in explaining the assignment of α in A: (*i*) how to compute a minimum cardinality set of atoms that is assumed to be false such that it is capable of explaining the assignment of α in A; and (*ii*) how to support sophisticated linguistic constructs such as choice rules and aggregates, which can be involved to explain the falsity of some atoms in easily understandable terms.

Our main contributions are the following:

- A notion of explanation in terms of directed acyclic graphs explains why an atom is (or is not) in an answer set in terms of easy-to-understand inferences originating from a hopefully minimum set

Submitted to:
ICLP 2023

of assumed false atoms (Section 3). Note that the explanation graph of an atom is restricted to the atoms involved in the relevant rules for the explaining atoms.

- A proof of existence for the explanations according to the given definition that guarantees the correctness of our implementation (Section 4).
- The implementation of an enhancement system, `xASP2` , for producing explanations powered by ASP and its empirical evaluation(Sections 5–6). `xASP2` tackles logic programs with different `clingo` constructs such as aggregates and constraints.

The supported fragment of ASP includes uninterpreted function symbols, common aggregation functions, comparison expressions, strong negation, constraints, normal rules, and choice rules. Aggregates are expected to be stratified, to not involve default negation, and to have a single atomic condition. Choice rules are expected to be unconditional, or otherwise to have exactly one conditional atom with a self-explanatory condition (as for example a range expression or an extensional predicate). Additionally, to ease the presentation, in Section 2 we only consider *sum* aggregates, and completely omit uninterpreted function symbols, comparison expressions, strong negation, and conditions in choice rules. To the best of our knowledge, this is the first explanation generation system that supports different `clingo` constructs such as aggregates and constraints.

2 Background

All sets and sequences considered in this paper are finite. Let $\mathbf{P}$, $\mathbf{C}$, $\mathbf{V}$ be fixed nonempty sets of *predicate names*, *constants* and *variables*. Predicates are associated with an *arity*, a non-negative integer. A *term* is any element in $\mathbf{C} \cup \mathbf{V}$. An *atom* is of the form $p(\bar{t})$, where $p \in \mathbf{P}$, and $\bar{t}$ 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.

An *aggregate* is of the form

$$sum\{t_a,\overline{t'} : p(\bar{t})\} \odot t_g \tag{1}$$

where $\odot$ is a binary comparison operator, $p \in \mathbf{P}$, $\bar{t}$ and $\overline{t'}$ are possibly empty sequences of terms, and t_a and t_g are terms.

A *choice* is of the form

$$t_1 \leq \{atoms\} \leq t_2 \tag{2}$$

where *atoms* is a possibly empty sequence of atoms, and t_1, t_2 are terms. Let $\perp$ be syntactic sugar for $1 \leq \{\} \leq 1$.

A *rule* is of the form

$$head \leftarrow body \tag{3}$$

where *head* is an atom or a choice, and *body* is a possibly empty sequence of literals and aggregates. For a rule r, let $H(r)$ denote the atom or choice in the head of r; let $B^{\Sigma}(r)$, $B^{+}(r)$ and $B^{-}(r)$ denote the sets of aggregates, positive and negative literals in the body of r; let $B(r)$ denote the set $B^{\Sigma}(r) \cup B^{+}(r) \cup B^{-}(r)$.

A variable X occurring in $B^{+}(r)$ is a *global variable*. Other variables occurring among the terms $\bar{t}$ of some aggregate in $B^{\Sigma}(r)$ of the form (1) are *local variables*. 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. Additionally, we assume that aggregates are stratified, that is, the *dependency graph* $\mathscr{G}_{\Pi}$ having a vertex for each predicate occurring in Π and an edge pq whenever there is $r \in \Pi$ with p occurring in $H(r)$ and q occurring in $B^{+}(r)$ or $B^{\Sigma}(r)$ is acyclic.

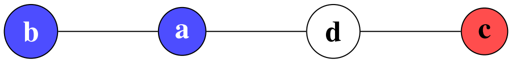

Figure 1: The undirected graph used as running example. Source vertices in blue, sink vertex in red.

Example 1. Given a connected undirected graph G encoded by predicate *edge/2*, source and sink nodes encoded by predicates *source/1* and *sink/1*, the following program assigns a direction to each edge so that source nodes can still reach all sink nodes:

$$
\begin{aligned}
&1 \leq \{arc(X,Y);\ arc(Y,X)\} \leq 1 \leftarrow edge(X,Y) && (4)\\
&reach(X,X) \leftarrow source(X) && (5)\\
&reach(X,Y) \leftarrow reach(X,Z),\ arc(Z,Y) && (6)\\
&\bot \leftarrow source(X),\ sink(Y),\ not\ reach(X,Y) && (7)
\end{aligned}
$$

If failures on the reachability condition are permitted up to a given threshold encoded by predicate *threshold/1*, the program comprising rules (4)–(6) and

$$
\begin{aligned}
&fail(X,Y) \leftarrow source(X),\ sink(Y),\ not\ reach(X,Y) && (8)\\
&\bot \leftarrow threshold(T), sum\{1,X,Y : fail(X,Y)\} > T && (9)
\end{aligned}
$$

can be used. Note that X and Y are local variables in rule (9), and all other variables are global. ■

A substitution σ is a partial function from variables to constants; the application of σ to an expression E is denoted by $E\sigma$. Let *instantiate*(Π) be the program obtained from rules of Π by substituting global variables with constants in $\mathbf{C}$, in all possible ways; note that local variables are still present in *instantiate*(Π). The Herbrand base of Π, denoted *base*(Π), is the set of ground atoms (i.e., atoms with no variables) occurring in *instantiate*(Π).

Example 2. Let Π_{run} comprise rules (4)–(6), (8)–(9) and the facts (i.e., rules with an empty body) $edge(a,b)$, $edge(a,d)$, $edge(d,c)$, $source(a)$, $source(b)$, $sink(c)$, and $threshold(0)$ (see Figure 1). Hence, *instantiate*
(Π_{run}) contains, among others, the rules

$$
\begin{aligned}
&1 \leq \{arc(a,b);\ arc(b,a)\} \leq 1 \leftarrow edge(a,b)\\
&\bot \leftarrow threshold(0), sum\{1,X,Y : fail(X,Y)\} > 0
\end{aligned}
$$

and $base(\Pi_{run})$ contains $fail(a,c), fail(b,c)$, and so on. ■

A *(two-valued) interpretation* is a set of ground atoms. 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 $I \models not\ p(\overline{c})$ if $p(\overline{c}) \notin I$; for an aggregate α of the form (1), the aggregate set of α w.r.t. I, denoted $aggset(\alpha,I)$, is $\{\langle t_a,\overline{t'}\rangle\sigma \mid p(\overline{t})\sigma \in I$, for some substitution $\sigma\}$, and $I \models \alpha$ if $(\sum_{\langle c_a,\overline{c'}\rangle \in aggset(\alpha,I)} c_a) \odot t_g$ is a true expression over integers; for a choice α of the form (2), $I \models \alpha$ if $t_1 \leq |I \cap atoms| \leq t_2$ is a true expression over integers; for a rule r with no global variables, $I \models B(r)$ 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 $r \in instantiate(\Pi)$.

For a rule r of the form (3) and an interpretation I, let $expand(r,I)$ be the set $\{p(\overline{c}) \leftarrow body \mid p(\overline{c}) \in I$ occurs in $H(r)\}$. The *reduct* of Π w.r.t. I is the program comprising the expanded rules of *instantiate*(Π) whose body is true w.r.t. I, that is, $reduct(\Pi,I) := \bigcup_{r \in instantiate(Pi),\ I \models B(r)} expand(r,I)$. An *answer set* of Π is an interpretation A such that $A \models \Pi$ and no $I \subset A$ satisfies $I \models reduct(\Pi,A)$.

Example 3. The only answer set A_{run} of program Π_{run} contains, among others, the atoms $arc(b,a)$, $arc(a,d)$, $arc(d,c)$, no other instance of $arc/2$, and no instance of $fail/2$. Hence, $A_{run} \models 1 \leq \{arc(a,b); arc(b,a)\} \leq 1$ and $A_{run} \not\models sum\{1,X,Y : fail(X,Y)\} > 0$. ■

A *three-valued interpretation* is a pair (L,U), where L,U are sets of ground atoms such that $L \subseteq U$; sets L and U, also denoted $(L,U)_1$ and $(L,U)_2$, are the lower and upper bounds on the true atoms, so 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 and aggregates with a truth value among $\mathbf{u}$, $\mathbf{t}$ and $\mathbf{f}$ as follows: $[\![\alpha]\!]_L^U = \mathbf{u}$ if α is a literal whose atom is $p(\overline{c})$ and $p(\overline{c}) \in U \setminus L$, or α is an aggregate of the form (1) and $aggset(\alpha, U \setminus L) \neq \emptyset$, or α is a choice of the form (2) and $(U \setminus L) \cap atoms \neq \emptyset$; $[\![\alpha]\!]_L^U = \mathbf{t}$ 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. For α being $sum\{1,X,Y : fail(X,Y)\} > 0$, $[\![\alpha]\!]_{\emptyset}^{\{fail(a,c)\}} = \mathbf{u}$, $[\![\alpha]\!]_{\{fail(a,c)\}}^{\{fail(a,c)\}} = \mathbf{t}$, and $[\![\alpha]\!]_{\emptyset}^{\emptyset} = \mathbf{f}$. ■

Mainstream ASP systems compute answer sets of a given program Π by applying several inference rules on (a subset of) $instantiate(\Pi)$, the most relevant ones for this work 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 $[\![B(r)]\!]_L^U = \mathbf{t}$. (Actually, if $H(r)$ is a choice of the form (2), inference by support additionally requires that $|atoms \cap U| = t_1$, that is, undefined atoms in $atoms \cap U$ are required to reach the bound t_1. Such extra condition is not relevant for our work, and will not be used, because our explanations aim at associating true atoms with rules with true bodies.) Atom $p(\overline{c})$ is *inferred false by lack of support* if each rule $r \in instantiate(\Pi)$ 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* $r \in instantiate(\Pi)$ 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 a choice rule* $r \in instantiate(\Pi)$ if $H(r)$ has the form (2), $p(\overline{c}) \in atoms$, $|atoms \cap L| \geq t_2$ and $[\![B(r)]\!]_L^U = \mathbf{t}$. 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 $r \in instantiate(\Pi)$ at least one of the following conditions holds: (i) no atom from X occurs in $H(r)$; (ii) $[\![B(r)]\!]_L^U = \mathbf{f}$; (iii) $B^+(r) \cap X \neq \emptyset$.

Example 5. Given the program $instantiate(\Pi_{run})$, and the three-valued interpretation $(\emptyset, base(\Pi_{run}))$, atom $edge(a,a)$ is inferred false by lack of support, atom $source(a)$ is inferred true by support, and the set $\{edge(a,a), arc(a,a)\}$ is unfounded. Given $(\{arc(d,c)\}, base(\Pi_{run}) \setminus \{reach(a,c)\})$, atom $reach(a,d)$ is inferred false by the constraint-like rule (6), and $arc(c,d)$ is inferred false by the choice rule (4). ■

3 Explanations

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. Note that $wf(\Pi,A)$ is computed as a preprocessing step.

Example 6. Given Π_{run} and A_{run} from Examples 2–3, the lower bound of $wf(\Pi_{run},A_{run})$ contains head atoms in Example 2, $arc(b,a)$, $arc(a,d)$, $arc(d,c)$, $reach(a,a)$, $reach(b,b)$, $reach(a,d)$, $reach(a,c)$, $reach(b,a)$, $reach(b,c)$, and $reach(b,d)$. The upper bound additionally contains $arc(a,b)$, $arc(d,a)$, $arc(c,d)$, and several instances of $reach/2$ and $fail/2$. ■

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

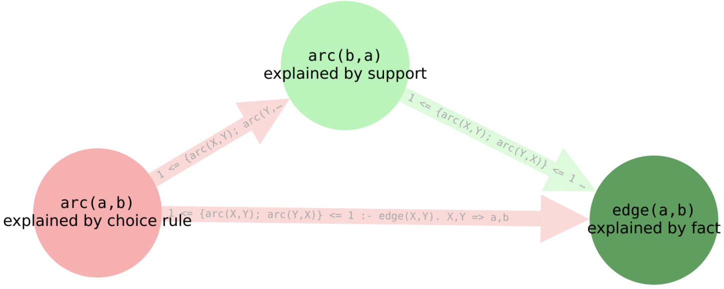
arc(b,a)
explained by support
arc(a,b)
explained by choice rule
edge(a,b)
explained by fact
1 <= {arc(X,Y); arc(Y,...
1 <= {arc(X,Y); arc(Y,X)} <= 1 ...
1 <= {arc(X,Y); arc(Y,X)} <= 1 :- edge(X,Y). X,Y => a,b

Figure 2: Induced DAG on the vertices reachable from $arc(a,b)$ for the minimal assumption set $\emptyset$ for Π_{run}.

support, constraint-like rules and choice 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 $(\emptyset, wf(\Pi,A)_2 \setminus X)$ terminates with A (in words, A is reconstructed from the false atoms of the well-founded derivation extended with X). 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 $X' \in AS(\Pi,A)$. (In other words, we prefer assumption sets not including the atom to explain. When all assumption sets include the atom to explain, we opt for the singleton comprising the atom to explain alone.) 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 can be checked that also $\emptyset \in AS(\Pi_{run}, A_{run}, \alpha)$, and it is indeed the only minimal assumption set in this case, for any atom in $base(\Pi_{run})$. ■

Given an assumption set X and an explaining derivation from $(\emptyset, wf(\Pi,A)_2 \setminus X)$, a directed acyclic graph (DAG) can be obtained as follows: The vertices of the graph are the atoms in $base(\Pi)$ and the aggregates occurring in $instantiate(\Pi)$. (The vertex $p(\bar{c})$ is also referred to as *not* $p(\bar{c})$.) Any aggregate of the form (1) is linked to instances of $p(\bar{t})$. 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 $B(r) \setminus \{\alpha\}$. Any atom α inferred false by a choice rule $r \in instantiate(\Pi)$ is linked to the atoms occurring in $H(r)$ that are true in A, and to the elements of $B(r)$. A portion of an example DAG is reported in Figure 2.

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.*

To prove 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* α the fact that $\alpha \in base(\Pi)$ is inferred false by lack of support, constraint-like rules and choice rules, which is the case when $[\![\alpha]\!]_L^U = \mathbf{u}$, and one of the following conditions holds: each rule $r \in instantiate(\Pi)$ 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 $[\![B(r) \setminus \{\alpha\}]\!]_L^U = \mathbf{t}$; there is $r \in \mathit{instantiate}(\Pi)$ with $H(r)$ of the form (2), $\alpha \in \mathit{atoms}$, $|\mathit{atoms} \cap L| \geq t_2$ and $[\![B(r)]\!]_L^U = \mathbf{t}$.

The *explaining derivation operator* D_Π is defined as

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

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. $\sqsubseteq$.*

Proof. For $(L,U) \sqsubseteq (L',U')$, we shall show that $D_\Pi(L,U) \sqsubseteq D_\Pi(L',U')$ holds. For $\alpha \in D_\Pi(L,U)_1 \setminus L$ such that $\alpha \notin L'$, we have $\Pi, L, U \vdash \alpha$, that is, there is $r \in \mathit{instantiate}(\Pi)$ such that α occurs in $H(r)$ and $[\![B(r)]\!]_L^U = \mathbf{t}$. As $(L,U) \sqsubseteq (L',U')$, we have that $[\![B(r)]\!]_{L'}^{U'} = \mathbf{t}$, that is, $\Pi, L', U' \vdash \alpha$ holds, and therefore $\alpha \in D_\Pi(L',U')_1 \setminus L$.

For $\alpha \in U \setminus D_\Pi(L,U)_2$ such that $\alpha \in U'$, we have $\Pi, L, U \vdash \mathit{not}\ \alpha$, and therefore we have three cases:

1. Each rule $r \in \mathit{instantiate}(\Pi)$ with α occurring in $H(r)$ is such that $[\![B(r)]\!]_L^U = \mathbf{f}$. As $(L,U) \sqsubseteq (L',U')$, $[\![B(r)]\!]_{L'}^{U'} = \mathbf{f}$ holds.
2. There is $r \in \mathit{instantiate}(\Pi)$ with $\alpha \in B^+(r)$, $[\![H(r)]\!]_L^U = \mathbf{f}$ and $[\![B(r) \setminus \{\alpha\}]\!]_L^U = \mathbf{t}$. As $(L,U) \sqsubseteq (L',U')$, $[\![H(r)]\!]_{L'}^{U'} = \mathbf{f}$ and $[\![B(r) \setminus \{\alpha\}]\!]_{L'}^{U'} = \mathbf{t}$.
3. There is $r \in \mathit{instantiate}(\Pi)$ with $H(r)$ of the form (2), $\alpha \in \mathit{atoms}$, $|\mathit{atoms} \cap L| \geq t_2$ and $[\![B(r)]\!]_L^U = \mathbf{t}$. As $(L,U) \sqsubseteq (L',U')$, $|\mathit{atoms} \cap L'| \geq t_2$ and $[\![B(r)]\!]_{L'}^{U'} = \mathbf{t}$.

In any case, $\Pi, L', U' \vdash \alpha$ holds, and therefore $\alpha \in U' \setminus D_\Pi(L',U')_2$. □

Lemma 2. *$L \subseteq A \subseteq U$ implies $D_\Pi(L,U)_1 \subseteq A \subseteq D_\Pi(L,U)_2$.*

Proof. For $\alpha \in D_\Pi(L,U)_1 \setminus L$ we have $\Pi, L, U \vdash \alpha$, that is, there is $r \in \mathit{instantiate}(\Pi)$ such that $[\![B(r)]\!]_L^U = \mathbf{t}$. Hence, $A \models B(r)$, and therefore $\mathit{expand}(r,A) \subseteq \mathit{reduct}(\Pi,A)$. In particular, $\alpha \leftarrow B(r)$ belongs to the reduct, and therefore $\alpha \in A$.

For $\alpha \in U \setminus D_\Pi(L,U)_2$ we have $\Pi, L, U \vdash \mathit{not}\ \alpha$ and we have to show that $\alpha \notin A$. Three cases:

1. Each rule $r \in \mathit{instantiate}(\Pi)$ with α occurring in $H(r)$ is such that $[\![B(r)]\!]_L^U = \mathbf{f}$.
2. There is $r \in \mathit{instantiate}(\Pi)$ with $\alpha \in B^+(r)$, $[\![H(r)]\!]_L^U = \mathbf{f}$ and $[\![B(r) \setminus \{\alpha\}]\!]_L^U = \mathbf{t}$.
3. There is $r \in \mathit{instantiate}(\Pi)$ with $H(r)$ of the form (2), $\alpha \in \mathit{atoms}$, $|\mathit{atoms} \cap L| \geq t_2$ and $[\![B(r)]\!]_L^U = \mathbf{t}$.

In the first case, $A \setminus \{\alpha\} \models \mathit{reduct}(\Pi,A)$, and therefore $A \setminus \{\alpha\} = A$ because A is an answer set of Π. In the other two cases, $\alpha \notin A$ because $A \models \Pi$ by assumption. □

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). Thus, the system eventually terminates in at most $|\mathit{base}(\Pi)|$ steps.

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

Proof. Let (L,U) be the explaining derivation from $(\emptyset, base(\Pi) \setminus A)$. Thanks to Lemma 2, it is sufficient to show that $p(\overline{c}) \in A$ implies $p(\overline{c}) \in L$. Let us consider a topological ordering $C_1, \ldots, C_n$ ($n \geq 1$) for the strongly connected components of $\mathscr{G}_\Pi$, and let $p \in C_i$. We use induction on i. Since $p(\overline{c}) \in A$, there must be $r \in reduct(\Pi, A)$ such that $H(r) = p(\overline{c})$ and $A \models B(r)$. Hence, $[\![B^-(r)]\!]_L^U = \mathbf{t}$. Moreover, $[\![B^\Sigma(r)]\!]_L^U = \mathbf{t}$, either because $i = 1$ and $B^\Sigma(r) = \emptyset$, or because of the induction hypothesis. Therefore, to have $\alpha \notin L$, it must be the case that $[\![B^+(r)]\!]_L^U \neq \mathbf{t}$ for all such rules, but in this case $L \models reduct(\Pi, A)$, a contradiction with the assumption that A is an answer set of Π. □

Given Lemma 3, the proof of Main Theorem is immediate by the definition of $MAS(\Pi, A, \alpha)$ as following:

Proof of Main Theorem. By definition, a minimal assumption set for Π, A and 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 $X' \in AS(\Pi, A)$. Lemma 3 guarantees the existence of an assumption set for Π and A. Existence of a minimal assumption set for Π, A and α is therefore guaranteed. □

5 Generation via Meta-Programming

By leveraging ASP systems, the concepts introduced in Section 3 can be computed. 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 weak constraints, uninterpreted functions, conditional literals and @-terms. The reader is referred to [5] 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 $serialize(\Pi, A, \alpha)$, defined next. Each atom $p(\overline{c})$ in $base(\Pi)$ 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 $wf(\Pi, A)$, 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 $instantiate(\Pi)$ is encoded by

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

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

```
  head($id(\overline{X})$,$p(\overline{t})$) :- 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 $p(\overline{t})$ occurring in $H(r)$, $p'(\overline{t'}) \in B^+(r)$ and $p''(\overline{t''}) \in B^{(}r)$; additionally, for each aggregate α of the form (1) in $B^\Sigma(r)$, the encoding includes

```
pos_body($id(\overline{X})$,$agg(\overline{X})$) :- rule($id(\overline{X})$).
aggregate($agg(\overline{X})$) :- rule($id(\overline{X})$).
true($agg(\overline{X})$) :- rule($id(\overline{X})$),
     #sum{$t_a$,$\overline{t'}$ : true($p(\overline{t})$)}$\odot t_g$.
false($agg(\overline{X})$):- rule($id(\overline{X})$), not true($agg(\overline{X})$).

rule($agg(\overline{X})$):- aggregate($agg(\overline{X})$),true($agg(\overline{X})$).
```

```
head($agg(\overline{X})$,$agg(\overline{X})$) :- rule($agg(\overline{X})$).
pos_body($agg(\overline{X})$,$p(\overline{t})$):- rule($agg(\overline{X})$),true($p(\overline{t})$).
neg_body($agg(\overline{X})$,$p(\overline{t})$):-rule($agg(\overline{X})$),false($p(\overline{t})$).

rule(($agg(\overline{X})$,$p(\overline{t})$)) :- aggregate($agg(\overline{X})$), false($agg(\overline{X})$), atom($p(\overline{t})$).
head(($agg(\overline{X})$,$p(\overline{t})$),$agg(\overline{X})$):- rule(($agg(\overline{X})$,$p(\overline{t})$)).
pos_body(($agg(\overline{X})$,$p(\overline{t})$),$p(\overline{t})$) :- rule(($agg(\overline{X})$,$p(\overline{t})$)), false($p(\overline{t})$).
neg_body(($agg(\overline{X})$,$p(\overline{t})$),$p(\overline{t})$) :- rule(($agg(\overline{X})$,$p(\overline{t})$)), true($p(\overline{t})$).
```

where agg is an identifier for α; finally, if $H(r)$ is a choice of the form (2), the encoding includes

```
  choice($id(\overline{X})$,$t_1$,$t_2$) :- rule($id(\overline{X})$).
```

Note that a true ground aggregate of the form (1) identified by $agg(\overline{c})$ is associated with a single rule whose body becomes true after all instances of $p(\overline{t})$ are assigned the truth value they have in the answer set A; on the other hand, a false aggregate is associated with one rule for each instance of $p(\overline{t})$, whose bodies becomes false when instances of $p(\overline{t})$ are assigned the truth value they have in the answer set A.

Example 8. Recall Π_{run} and A_{run} from Examples 2–3. The ASP program $serialize(\Pi_{run}, A, arc(a,b))$ includes

```
atom(edge(a,b)).  atom(arc(b,a)).  atom (arc(a,b)).  explain(arc(a,b)).
true(edge(a,b)).  true(arc(b,a)).  false(arc(a,b)).

rule(r4(X,Y)) :- atom(edge(X,Y)).
choice(r4(X,Y),1,1) :- rule(r4(X,Y)).
head(r4(X,Y), arc(X,Y)) :- rule(r4(X,Y)).
head(r4(X,Y), arc(Y,X)) :- rule(r4(X,Y)).
pos_body(r4(X,Y), edge(X,Y)):- rule(r4(X,Y)).

aggregate(agg1(T)) :- rule(r9(T)).
true(agg1(T)) :- rule(r9(T)), #sum{1,X,Y : true(fail(X,Y))} > T.
```

and several other rules. The answer set of $serialize(\Pi_{run}, A, arc(a,b))$ includes, among other atoms, `aggregate(agg1(0))` and `false(agg1(0))`. ■

The ASP program Π_{MAS} reported in Figure 3, coupled with a fact for each atom in the answer set of $serialize(\blacksquare, A, \alpha)$, has optimal answer sets corresponding to cardinality-minimal elements in $MAS(\Pi, A, \alpha)$. 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 $wf(\Pi, A) \setminus X$, 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 9. Let Π_S have a fact for each atom in the answer set of $serialize(\Pi_{run}, A_{run}, arc(a,b))$. $\Pi_{MAS} \cup \Pi_S$ generates the empty assumption set. $\Pi_{EXP} \cup \Pi_S \cup \emptyset$ generates an explaining derivation, for example one including $explained\_by(edge(a,b), (support, r6))$, $explained\_by(arc(b,a), (support, r1(a,b)))$ and $explained\_by(arc(a,b), (choice\_rule, r1(a,b)))$. Let Π_E have a fact for each instance of $explained\_by/3$

```
{assume_false(Atom)} :- false(Atom), not aggregate(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), not aggregate(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).
explained_head(Rule) :- rule(Rule); has_explanation(HAtom) : head(Rule,HAtom).
false_head(Rule) :- explained_head(Rule), not choice(Rule,_,_);
   false(HAtom) : head(Rule,HAtom).
false_head(Rule) :- explained_head(Rule), choice(Rule, LowerBound, UpperBound);
   not LowerBound <= #count{HAtom' : head(Rule,HAtom'), true(HAtom')} <= UpperBound.

{explained_by(Atom, (choice_rule, Rule))} :- false(Atom);
   head(Rule,Atom), choice(Rule, LowerBound, UpperBound);
   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);
   #count{HAtom : head(Rule, HAtom), true(HAtom), has_explanation(HAtom)} = UpperBound.
```

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

in the explaining derivation. $\Pi_{DAG} \cup \Pi_S \cup \Pi_E$ generates a DAG, for example one including $link(arc(b,a), edge(a,b))$, $link(arc(a,b), arc(b,a))$ and $link(arc(a,b), edge(a,b))$. ■

6 Implementation and Experiment

We deployed an XAI system for ASP named `xASP2` , which is powered by the `clingo python api` [8]. By taking an ASP program Π, one of its answer sets A, and an atom α as input, `xASP2` is capable of producing minimal assumption sets, explaining derivations, and DAGs as output to assist the user in determining 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 $instantiate(\Pi)$ and $base(\Pi)$. In a nutshell, ground atoms provided by the user, $A \cup \{\alpha\}$, are part of $base(\Pi)$ and used to instantiate rules of Π (by matching positive body literals), which in turn may extend $base(\Pi)$ with other ground atoms occurring in the instantiated rules; possibly, some atoms of $base(\Pi)$ of particular interest can be explicitly provided by the user. Aggregates are also processed automatically by means of an ASP program, and so is the computation of false atoms in 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 opti-

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

{link(Atom, A) : pos_body(Rule,A), false(A), explained_by(I,A,_), I < Index;
 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).

link(Atom,A) :- explained_by(_, Atom, (required_to_falsify_body, Rule)); head(Rule,A).
link(At,A) :- explained_by(_,At,(required_to_falsify_body, Rule)); pos_body(Rule,A), A!=At.
link(Atom,A) :- explained_by(_,Atom,(required_to_falsify_body, Rule)); neg_body(Rule,A).

link(Atom, HAtom) :- explained_by(_,Atom,(choice_rule, Rule)); head(Rule,HAtom), true(HAtom).
link(Atom, BAtom) :- explained_by(_, Atom, (choice_rule, Rule)); pos_body(Rule, BAtom).
link(Atom, BAtom) :- explained_by(_, Atom, (choice_rule, Rule)); neg_body(Rule, BAtom).
```

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

mization, the explaining derivation is shrunk to the atoms reachable from α, utilizing 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. 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 is 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.

Table 1: The action preconditions and effects in Blockworlds problem

Action	**Precondition**	**Effects**
$stack(X,Y)$ - stack block X is on block Y	Block Y is clear The agent holds the block X	X is clear X is on Y Y is no longer clear The agent does not hold anything
$unstack(X,Y)$ - unstack block X is on block Y	X is clear X is on Y The agent does not hold anything	The agent holds the block X Y becomes clear X is not clear
$pickup(X)$ - pickup block X from the table	X is clear X is on the table the agent does not hold anything	The agent holds the block X X is no longer on the table and is not clear
$putdown(X)$ - put down block X onto the table	The agent holds the block X	X is clear X is on the table the agent does not hold anything

`xASP2` also has the ability to handle explainable planning, meaning it can generate an explanation

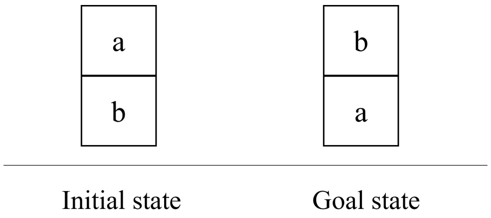

Figure 5: The initial and goal states of Blocksworld.

```
1 h(X,T+1) :- action(action(A)),occurs(A,T), postcondition(action(A),
      effect(unconditional),X,value(X,true)).
2 -h(X,T+1) :- action(action(A)),occurs(A,T), postcondition(action(A),
      effect(unconditional),X,value(X,false)).
3 h(X,T+1) :- h(X,T), not -h(X,T+1).
4 -h(X,T+1) :- -h(X,T), not h(X,T+1).
5 non_exec(A,T) :- action(action(A)), not h(X,T), precondition(action(A),X,value(X, true)).
6 non_exec(A,T) :- action(action(A)), not -h(X,T), precondition(action(A),X,value(X, false)).
7 :- action(action(A)),occurs(A,T), non_exec(A,T).
```

Figure 6: ASP program for reasoning about effects of actions [12]

graph showing why a particular action cannot take place at a certain time. To demonstrate this capability, we will use a popular problem known as Blocksworld. The initial state (left) and goal state (right) of the problem are shown in Figure 5. Five fluents are $on(X,Y)$ - block X is on block Y, $onTable(X)$ - block X is on the table, $clear(X)$ - block X is clear, $holding(X)$ - the agent holds the block X, and $handEmpty$ - the agent does not hold anything. Four differnt actions are *stack*, *unstack*, *pickup* and *putdown*. The domain description of the problem is shown in Table 1 in which the predictions and effects of four actions are presented.

The rules for reasoning about effects of actions, action generation and goal enforcement [12] are utilized as programming input in `xASP2` . Figure 6 shows the ASP program for reasoning about the effects of actions in which an action occurs only when its preconditions are true and then its effects are true in the next time step. Specifically, lines 5 and 6 are used to define states in which an action cannot be executed, and constraint is employed to prevent non-executable actions from occurring (line 7).

For the problem described in Figure 5, executing the actions of $unstack(a,b)$, $putdow(a)$, $pickup(b)$ and $stack(b,a)$ at times 0, 1, 2, and 3 respectively constitutes the optimal plan (assuming time starts at 0). However, if users are in a rush and want to put down block a on the table at time 0, as represented by the atom $occurs((\text{"}putdown\text{"}, constant(\text{"}a\text{"})), 0)$, they will encounter a false occurrence of the action $putdown(a)$. Figure 6 shows that atom $occurs((\text{"}putdown\text{"}, constant(\text{"}a\text{"})), 0)$ is false because the constraint rule prevents its execution and the prediction of the action holding block a is invalid/false.

7 Related Work

Our work, as stated in the introduction, is situated within the realm of XAI and can be used as a debugging tool to provide explanations for an unexpected result. For instance, if an element α is false in all answer sets of a program Π despite user expectations, our system can help identify which rules are contributing to this anomalous behavior. Thus, in this section, we will explore both debugging tools for ASP and cutting-edge XAI systems designed for ASP.

In Table 2, a summary of the compared features is presented. The features include as follows:

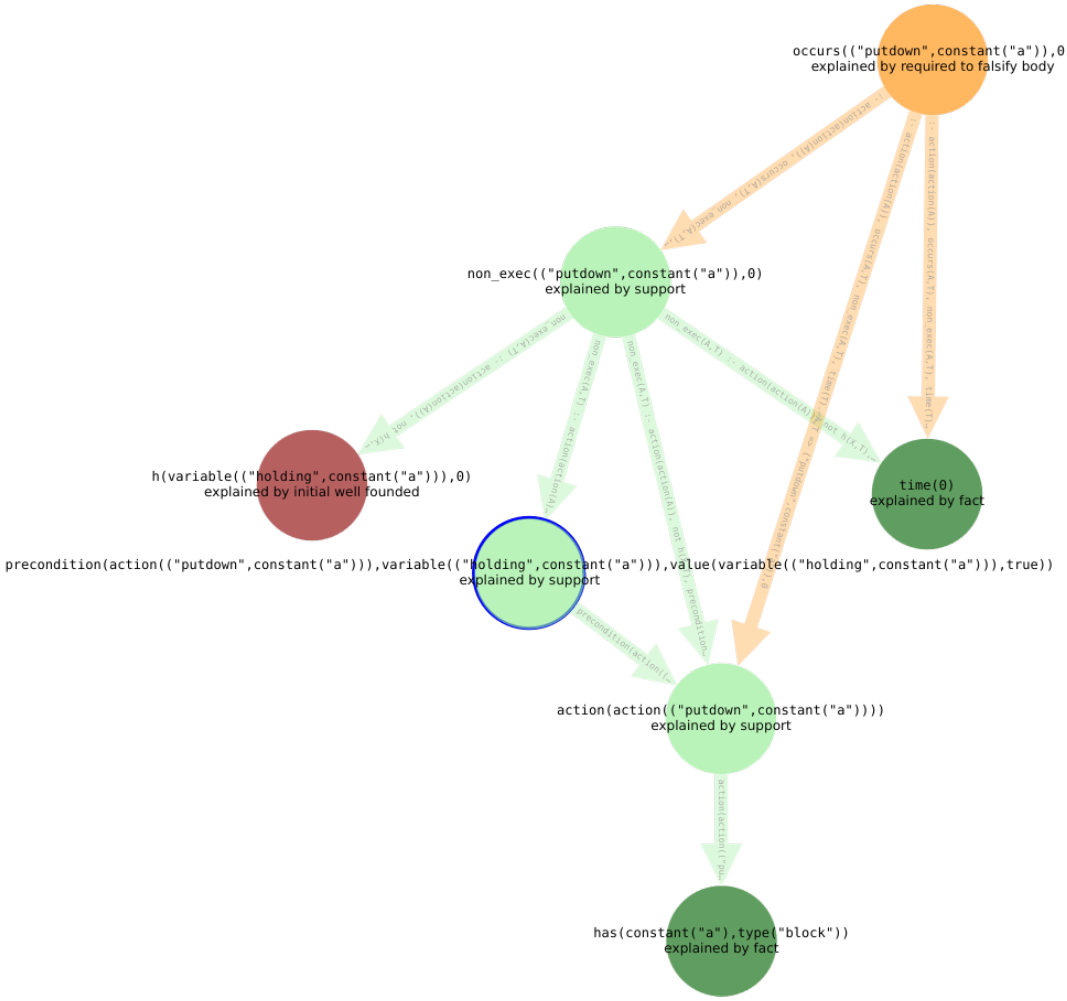
occurs(("putdown",constant("a")),0)
explained by required to falsify body
non_exec(("putdown",constant("a")),0)
explained by support
h(variable(("holding",constant("a"))),0)
explained by initial well founded
time(0)
explained by fact
precondition(action(("putdown",constant("a"))),variable(("holding",constant("a"))),value(variable(("holding",constant("a"))),true))
explained by support
action(action(("putdown",constant("a"))))
explained by support
has(constant("a"),type("block"))
explained by fact

Figure 7: The DAG for atom $occurs((\text{"}putdown\text{"}, constant(\text{"}a\text{"})), 0)$.

whether the explanation is guaranteed to be acyclic; the capability to handle the input program with aggregates and constraints; the ability to provide an explanation when the query atom can be false in the answer set; and whether the system is available for experimentation. As can be seen from Table 2, our system is capable of providing explanations for false atoms and does not lead to cyclic argumentation in the explanation. xASP2 is the only system that tackles a program that includes both aggregates and constraints. It is worth noting that the topic of aggregates is addressed in another approach [11], even though no system implementing this approach is mentioned or available.

8 Conclusion

We have developed and implemented xASP2 , an XAI system that targets the ASP language and is powered by ASP engines. Our approach to explaining why an atom is true/false in an answer set involves deriving easy-to-understand inferences originating from a hopefully small set of atoms assumed false. xASP2 has the ability to support different clingo constructs such as aggregates and constraints. It produces an explanation as a DAG with the atom to be explained as the root and takes a few seconds to compute the explanation in our test cases. Further investigation to include ASP's other linguistic

Table 2: Summary of compared features

System (if any) and reference	**Acyclic explanation**	**Linguistic extentions**	**Explanation for false atoms**	**System availability**
`s(CASP)` [1]	Yes	Constraints	Yes	Yes
`ASPeRiX` [2]	Yes	Constraints	Yes	Yes
`spock` [3]	Yes	Constraints	No	Yes
`xclingo` [4]	Yes	None	No	Yes
`DWASP` [6]	Yes	Constraints	No	Yes
[11]	No	Aggregates	Yes	No
`Visual-DLV` [15]	Yes	Constraints	No	Yes
[16]	No	None	Yes	No
`LABAS` [17]	No	None	Yes	Yes
exp(ASP^{c}) [18]	No	Constraints	Yes	Yes
[20]	Yes	None	Yes	No
`xASP2`	Yes	Aggregates and Constraints	Yes	Yes

constructs such as conditional literals, beyond those currently supported, present in the future work.

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] Joaquín Arias, Manuel Carro, Zhuo Chen & Gopal Gupta (2020): *Justifications for Goal-Directed Constraint Answer Set Programming*. *Electronic Proceedings in Theoretical Computer Science* 325, p. 59–72, doi:10.4204/EPTCS.325.12.

[2] Christopher Béatrix, Claire Lefèvre, Laurent Garcia & Igor Stéphan (2016): *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, pp. 6:1–6:15, doi:10.4230/OASIcs.ICLP.2016.6.

[3] Martin Brain, Martin Gebser, Jörg Pührer, Torsten Schaub, Hans Tompits & Stefan Woltran (2007): *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)*, pp. 71–85.

[4] Pedro Cabalar, Jorge Fandinno & Brais Muñiz (2020): *A System for Explainable Answer Set Programming*. *Electronic Proceedings in Theoretical Computer Science* 325, p. 124–136, doi:10.4204/EPTCS.325.19.

[5] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca & Torsten Schaub (2020): *ASP-Core-2 Input Language Format*. Theory Pract. Log. Program. 20(2), pp. 294–309, doi:10.1017/s1471068419000450.

[6] Carmine Dodaro, Philip Gasteiger, Kristian Reale, Francesco Ricca & Konstantin Schekotihin (2019): *Debugging non-ground ASP programs: Technique and graphical tools*. Theory and Practice of Logic Programming 19(2), pp. 290–316, doi:10.1017/S1471068418000492.

[7] M. Gelfond & V. Lifschitz (1990): *Logic programs with classical negation*. In D. Warren & Peter Szeredi, editors: Logic Programming: Proc. of the Seventh International Conference, pp. 579–597.

[8] Roland Kaminski, Javier Romero, Torsten Schaub & Philipp Wanko (2023): *How to Build Your Own ASP-based System?!* Theory and Practice of Logic Programming 23(1), p. 299–361, doi:10.1017/S1471068421000508.

[9] Fang Li, Huaduo Wang, Kinjal Basu, Elmer Salazar & Gopal Gupta (2021): *DiscASP: A Graph-based ASP System for Finding Relevant Consistent Concepts with Applications to Conversational Socialbots*. In: Proceedings 37th International Conference on Logic Programming (Technical Communications), ICLP Technical Communications 2021, Porto (virtual event), 20-27th September 2021, EPTCS 345, pp. 205–218, doi:10.4204/EPTCS.345.35.

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

[11] Simon Marynissen, Jesse Heyninck, Bart Bogaerts & Marc Denecker (2022): *On nested justification systems (full version)*. CoRR abs/2205.04541, doi:10.48550/arXiv.2205.04541.

[12] Van Nguyen, Stylianos Loukas Vasileiou, Tran Cao Son & William Yeoh (2020): *Explainable planning using answer set programming*. In: Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning, pp. 662–666, doi:10.24963/kr.2020/66.

[13] I. Niemelä (1999): *Logic programming with stable model semantics as a constraint programming paradigm*. Annals of Mathematics and Artificial Intelligence 25(3,4), pp. 241–273, doi:10.1023/A:1018930122475.

[14] Nikolay Pelov, Marc Denecker & Maurice Bruynooghe (2007): *Well-founded and stable semantics of logic programs with aggregates*. Theory Pract. Log. Program. 7(3), pp. 301–353, doi:10.1017/S1471068406002973.

[15] Simona Perri, Francesco Ricca, Giorgio Terracina, D Cianni & P Veltri (2007): *An integrated graphic tool for developing and testing DLV programs*. In: Proceedings of the Workshop on Software Engineering for Answer Set Programming (SEA'07), pp. 86–100.

[16] Enrico Pontelli, Tran Cao Son & Omar Elkhatib (2009): *Justifications for logic programs under answer set semantics*. Theory and Practice of Logic Programming 9(1), pp. 1–56, doi:10.1017/S1471068408003633.

[17] Claudia Schulz & Francesca Toni (2016): *Justifying answer sets using argumentation*. Theory and Practice of Logic Programming 16(1), pp. 59–110, doi:10.1017/S1471068414000702.

[18] Ly Ly Trieu, Tran Cao Son & Marcello Balduccini (2021): *exp(aspc): explaining asp programs with choice atoms and constraint rules*. Electronic Proceedings in Theoretical Computer Science, doi:10.4204/EPTCS.345.28.

[19] Ly Ly Trieu, Tran Cao Son & Marcello Balduccini (2022): *xASP: An Explanation Generation System for Answer Set Programming*. In: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, pp. 363–369, doi:10.1007/978-3-031-15707-3_28.

[20] Carlos Viegas Damásio, Anastasia Analyti & Grigoris Antoniou (2013): *Justifications for logic programming*. In: Logic Programming and Nonmonotonic Reasoning: 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proc. 12, Springer, pp. 530–542, doi:10.1007/978-3-642-40564-8_53.